Programmer's Guide - English

FUJITSU

openFT V11.0 for Unix Systems and Windows Systems

C Program Interface

Edition August 2010

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion of this manual. Your feedback helps us optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to manuals@ts.fujitsu.com.

Certified documentation according to DIN EN ISO 9001:2008

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Copyright © Fujitsu Technology Solutions GmbH 2010.

All rights reserved.
Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

This manual is printed on paper treated with chlorine-free bleach.

Contents

1 Changes to the C program interface 5

1.1 Overview 6

1.2 Programming rules 7
1.2.1 File transfer 7
1.2.2 File management 12
1.2.3 Specifications concerning the remote system 12
1.2.4 Error handling 15
1.2.5 Version of the program interface 16

2 Creating and using programs 17

2.1 Translating and linking under Windows systems 17

2.2 Translating and linking under Unix systems 18

2.3 Notes for program use 18

3 Description of the C functions 19

3.1 ft_cancel - Cancel asynchronous request 20

3.2 ft_close - Close session 22

3.3 ft_credir - Create directory in remote system 23

3.4 ft_delete - Delete file or directory in the remote system . . . 25

3.5 ft_open - Open session 28

3.6 ft_properties - Determine properties of the program interface 30

3.7 ft_reqlist - Determine request not yet terminated 33

3.8 ft_reqstat - Determine the status of a request 35

3.9 ft_reqterm - Terminate request 38

3.10 ft_show - Determine attributes of a file or directory 40

3.11 ft_showdir - Determine the attributes of all file in a directory 46

3.12 ft_transfer - Transfer file 53

4 Error codes 67

4.1 Internal errors 68

4.2 Parameter errors 69

4.3 Sequence errors 76

5 Sample programs 79

Sample 1: Asynchronous file transfer of a file 79
Sample 2: Several file transfer requests with follow-up processing 94
Sample 3: Display contents of a directory in a remote system . 109

Index 119

1 Changes to the C program interface

You can use the C program interface to incorporate the functionality of openFT in your own C programs:

– synchronous file transmission

– asynchronous file transfer

– managing and deleting asynchronous file transfer requests

– determining file attributes in the remote system

– deleting files or directories in the remote system

– creating directories in the remote system

These functions which are available to the openFT user can be used in C programs to automate sequences. The program interface naturally also provides monitoring and error handling mechanisms.
In addition, the program interface has a function call which you can use determine the properties of the program interface. You can use this call to check the properties and thus render your programs insensitive to changes in later versions.

Under Windows, the program interface supports **multithreading**, i.e. all program interface calls are threadsafe.

1.1 Overview

The following overview is useful for quick orientation with respect to which C program calls are available for which tasks. The corresponding FT commands which the user can work with on the shell level are indicated in brackets (see online help).

File transfer function

ft_transfer	Transfer file (*ft* oder *ncopy*)

Functions for managing asynchronous file transfer requests

ft_open	Open session
ft_close	Close session
ft_reqlist	Determine requests that have not been completed
ft_reqstat	Determine request status
ft_reqterm	Terminate request
ft_cancel	Abort request (*ftcanr*)

File management functions

ft_show	Determine the attributes of a file or directory in the remote system (ftshw)
ft_showdir	Determine all file attributes of a directory in the remote system (*ftshw -d*)
ft_delete	Delete a file or directory in the remote system (delete file: *ftdel*; delete directory: *ftdeldir*)
ft_credir	Create a directory in the remote system (*ftcredir*)

Function for querying properties of the program interface

ft_properties	Determine properties of the program interfacen

1.2 Programming rules

This section describes the points which you must observe when creating programs for the program interface of openFT.

1.2.1 File transfer

Synchronous transfer

For synchronous file transfer, use the function *ft_transfer()*. In the parameter list, the parameter *synchron* **must** contain the value FT_SYNC. The control is not returned to the program until file transfer is completed. You can use the return values to determine whether file transfer has been successful.

Asynchronous file transfer

Several functions are necessary in order to perform synchronous file transfer. They result from the fact that, with asynchronous file transfer requests are issued, stored in the request queue and possibly not executed until later. The requests must be administered, and monitored for successful completion. It is therefore only possible to transfer files asynchronously within „sessions“.

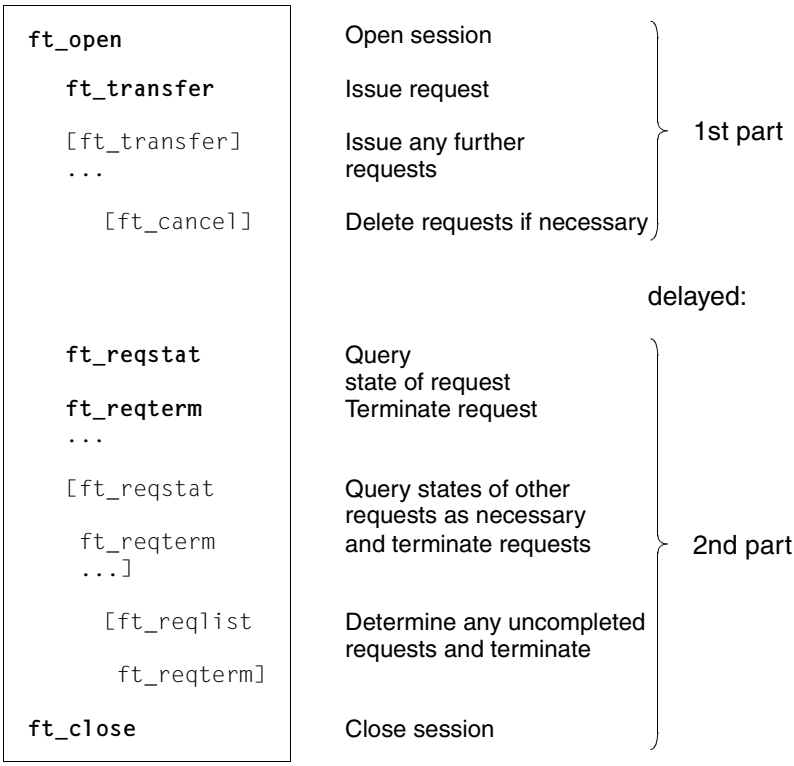
A program for asynchronous file transfer is made of two parts:

– In the first part, you open a session. Further, you issue one or more file transfer requests. If necessary, you can delete file transfer requests.

  openFT executes the request itself at the next possible opportunity.

– In the second part, you query the status of the request later and terminate the request on successful completion. If necessary, you can determine which requests have not been completed and terminate these as appropriate. Then you can close the session.

Schematic of the program structure:



The following function calls are absolutely necessary for asynchronous file transfer:

1. *ft_open()*

   The function *ft_open()* opens a session. The result of *ft_open()* is a session number (session identification) which uniquely identifies the session. This session number must be specified as parameter for function calls within the same session.

   When you open a session, you must assign an existing directory as working directory. In this working directory, the files are stored with management information about the existing file transfer requests.

You may assign the same working directory to several different sessions. This brings the advantage that you can administer requests from various sessions together.

You can conduct several sessions in parallel in one program. With *ft_open()*, however, you can only open more than one session simultaneously if each of the parallel sessions is assigned a different working directory.

2. *ft_transfer()*

   The *ft_transfer()* is used to issued an asynchronous file transfer request. The parameter *synchron* in the parameter list **must** contain the value `FT_ASYNC`. You may issue several asynchronous requests in succession in one session.

   When the request has been successfully entered in the request queue, you are returned a request ID as the result of *ft_transfer()*. The request ID uniquely identifies the request. This request ID is valid, even beyond the session, until the request is terminated with the function *ft_reqterm()*.

   When the request is present in the request queue, you do not have to take care of the request execution. openFT executes the asynchronous request at the earliest possible time. If, for example, a partner is not available at the moment, openFT keeps trying to execute the request. The request is thus held in the request queue until it has been completed.

3. *ft_reqstat()*

   You can use the function *ft_reqstat()* to determine whether or not the file transfer has been successfully completed. As asynchronous file transfer is not carried out immediately, you should delay the use of the function *ft_reqstat()* and repeat the status query. When the request is completed, the parameter *status* is assigned the value `FT_STATT`, if terminated, it is assigned the value `FT_STATA`.

4. *ft_reqterm()*

   You **must** terminate the request using the function *ft_reqterm()*. This function deletes the request ID of the request and also the file in which the relevant information about the file transfer request is stored. Resources which are no longer required are released.

   The management file has the name `mf.`*`Request-ID`* and is located in the directory indicated as *workdir* parameter with the function call *ft_open()*.

The request IDs of requests which have not been completed are retained, even after the session in which the requests were issued has been closed These requests can be completed at a later time by referencing the associated request ID, if the current session is assigned to the same working directory as the session in which the request was originally issued.

5. *ft_close()*

   You can use the function *ft_close()* to close the session.

HOME directory

If absolute file or directory names were not specified in the remote Unix or Windows systems then the user's HOME directory in the remote system is of importance. Relative path names always refer to the HOME directory of the corresponding user unless a definition to the contrary has been made by an FTAC profile.

The following applies to the HOME directory:

- In Unix systems, the HOME directory is the directory which is opened for the user after login.
- In Windows systems, a user's HOME directory for openFT requests is the directory entered in user administration for this user.

  If no directory is entered for the user in user administration then the user's profile path is used as the HOME directory. The profile path is *\Documents and Settings\User* where *User* does not have to be identical to the name of the user.

  If it is also not possible to determine the user's profile path, then the openFT home directory is created in the *\Documents and Settings* directory and access rights are granted in full to SYSTEM, administrators and the corresponding user.
  In this case, the name of the home directory created by openFT is determined as follows:
  – Local user ID: *UserID.Computer-name*
  – Global user ID (domain\user ID): *UserID.Domain*

  i The previously used HOME directory under *..\Program Files\openFT* is no longer supported. Home directories created by openFT are not deleted when openFT is uninstalled.

Managing asynchronous requests

Further functions are available for managing asynchronous requests:

– *ft_cancel()*

  You can use the function *ft_cancel()* to cancel asynchronous requests which are in the course of being processed or which are waiting to be processed in the request queue request queue.

  This function is expedient only when the program to be created has a user interface and the user has options for intervening. For example, you could imaging a program which displays waiting file transfer requests to the user (ft_reqstat (status=FT_STATW)) and allow him or her to abort these requests (e.g. by pressing the DEL key).

  Another application is when file transfer request have issued by mistake and should now be deleted.

  With the function *ft_cancel()*, you can only cancel requests which are present in the request queue and which have a request ID. If the request was issued in another session, the current session must be assigned the same working directory as for the session in which the request was issued. If the *ft_transfer()* function returns the value 0, the request could not be entered into the request queue. These unsuccessful attempts to enter requests terminate with an error message.

  Requests which you cancel with *ft_cancel()*, **must** be terminated with *ft_reqterm()* in order for the associated request ID to be deleted.

– *ft_reqlist()*

  All transfer requests must be completed so that associated request IDs and management files can be deleted and resources that are not required released. With the function *ft_reqlist(),* you can determine requests that are not yet completed and then terminate them with the function *ft_reqterm()*.

  With the function *ft_reqlist(),* you determine requests from all sessions that are not yet completed and which have been assigned the same working directory as the current session. Please note that not all request which have not be completed are determined, but only those from sessions with the same working directory.

1.2.2 File management

Two functions are available for determining the file attributes in the remote system:

– With the function *ft_show()*, you can list the attributes of **one** file.

– With the function *ft_showdir()*, you can list the attributes of **all** files in a directory.

In order to delete a file or a directory in the remote system, use the function *ft_delete()*.

1.2.3 Specifications concerning the remote system

All functions that access a remote system must identify the remote system and make the transfer admission known. The file structure *ft_admission* is used.

In Windows, the *ft_admission* structure is defined in the header file *...\openFT\include\ftapi.h*.

In Unix systems, the *ft_admission* structure is defined in the header file */usr/include/ftapi.h*.

ft_admission

The structure *ft_admission* is set up as follows:

```
struct ft_admission
{
   char  *remsys;         /* input */
   char  *remadmis;       /* input */
   char  *remaccount;     /* input */
   char  *rempasswd;      /* input */
};
```

The fields of the structure *ft_admission* have the following meaning.

remsys

Name of the partner system in the partner list or address of the partner system.

The address of the partner system is specified in the following form:

[protocol://]host[:[port].[tsel].[ssel].[psel]]

protocol
Protocol stack via which the partner is addressed.
Possible values:

openft (openFT protocol), default value

ftam (FTAM protocol)

ftp (ftp protocol)

host Internet host name, IP address or GLOBAL NAME from the TNS, mandatory parameter. Format of the IP addresses (example): `%ip111.222.123.234` (IPv4) or `%ip6[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]` (IPv6)
The square brackets [..] must be specified wth IPv6.

port Port number for TCP/IP connection, optional.

tsel Transport selector (only openFT and FTAM protocol), optional.

ssel Session selector for FTAM connection, optional.

psel Presentation selector for FTAM connection, optional.

For further details on addressing partner systems see the online help system or, in the case of Unix systems, refer to the openFT User Manual.

remadmis
Either a login name in the remote system or an FTAC transfer admission.

remaccount
Account number in the remote system

rempasswd
Password in the remote system

Depending on the particular type of openFT partner system, the following entries are necessary:

With BS2000:
remadmis, if the remote system uses the FTAC transfer admission; otherwise: *remadmis*, *remaccount* and, if a password is assigned, *rempasswd*

With Unix systems:
remadmis, if the remote system uses the FTAC transfer admission; otherwise: *remadmis* and, if a password is assigned, *rempasswd*

With Windows systems:
remadmis, if the remote system uses the FTAC transfer admission; otherwise: *remadmis*, if a login name is assigned, and *rempasswd*, if a password is assigned

With OS/390 and z/OS:
remadmis, *remaccount* and, if a password is assigned, *rempasswd*

With FTAM partner systems, for which no product of the openFT product range is in use:
remadmis, if an account number is assigned, *remaccount* and, if a password is assigned, *rempasswd*

For other partner systems:
corresponding to the conventions of the particular partner system

All fields not specified must contain the value 0.

1.2.4 Error handling

All function calls end with a return message. The return value indicates successful completion or informs the user globally that an error has occurred. You can obtain detailed information by calling the a function with optional parameter *errorinfo*. Immediately after an error has occurred, the structure *ft_err* contains error messages with which you can program corresponding error handling procedures.

In Windows, the *ft_err* structure is defined in the header file *...\openFT\include\ftapi.h*.

In Unix systems, the *ft_err* structure is defined in the header file */usr/include/ftapi.h*.

ft_err

The structure *ft_err* is set up as follows:

```
struct ft_err
{
   long  main;           /* output */
   long  detail;         /* output */
   long  additional;     /* output */
};
```

The fields of the structure *ft_err* have the following meanings:

main
: contains the error class, e.g. parameter error, internal error

detail
: describes the error, e.g. invalid parameter value

additional
: contains additional error information, e.g. which parameter is invalid

The error codes are described in the chapter “Error codes” on page 67.

1.2.5 Version of the program interface

You can use the function call *ft_properties()* to determine the version of the openFT program interface, as well as important version-specific system values. With this function, you ensure the executability of future versions of openFT - even without recompilation. This function call is above all important when you use programs that are to run with different versions of the program interface.

ft_options

The *ft_credir()* function introduced in version 2 of the openFT program interface and the extended file structures can only be used if the *options* parameter is specified for the corresponding functions.

The *ft_options* structure is constructed as follows:

```
struct ft_options
{
   int ftoptsvers;   /* input */
   int ftapivers;    /* input */
};
```

The fields in the structure have the following meaning:

ftoptsvers
: Version of the data structure.
The value `FT_OPTSV1` must be entered for *ftoptsvers*.

fftapivers
: Specifies the version of the program interface:

  `FT_APIV2`
  : The openFT message number (*ft_err.detail*=`FTED_FTMSG`) specified in the *additional* parameter adheres to the new message number schema which was introduced in openFT V10.

2 Creating and using programs

Include file

All C programs which use the program interface of openFT must contain the following line:

```
#include <ftapi.h>
```

The data types and function prototypes are defined in this include file.

In Windows, this include file is located in in the *openFT\include* sub-directory of the openFT installation directory.

2.1 Translating and linking under Windows systems

The edited C program must be compiled and linked with the import library *ftapi.lib*. This library is located in the *openFT\lib* directory of the openFT installation directory.

During runtime, the library *ftapi.dll* is also dynamically loaded from the directory *..\openFT\bin*.

ftapi.lib and *ftapi.dll* were created using Microsoft Visual Studio .NET 2003.

64-bit support in Windows systems

This version also includes a 64-bit DLL with the name *ftapi64.dll* for the Windows x64 systems as well as for the Windows Itanium systems.

During openFT installation, the *ftapi64.dll* variant corresponding to the employed operating system is automatically installed in the directory *..\openFT\bin*.

The associated import library *ftapi64.lib* is located in the directory *..\openFT\lib\x64* in the case of Windows x64 systems and the directory *..\openFT\lib\ia64* in the case of Windows Itanium systems.

2.2 Translating and linking under Unix systems

The edited C program must be compiled, and the openFT functions must be linked from the openFT library. Call the C compiler with the *-lftapi* option.

The following options must also be specified on some systems:

LINUX	`-lcrypt`
AIX	-Wl, -brtl
HP (Itanium 64bit)	+DD64

Note that under HP-UX, the C compiler must always be called in ANSI mode.

2.3 Notes for program use

Information relating to asynchronous file transfer requests are stored in files with the name `mf`.*Request-ID* in the directory indicated in the *workdir* parameter of the *ft_open()* function call. These data are deleted when you terminate requests with the function call *ft_reqterm()*.

3 Description of the C functions

Representation

The following conventions are used to represent the functions:

`stenographic`
: for shell commands, function calls, programs and subprograms, as well as for constant values in plain text.

italics
: for function names and parameters

In the syntactic representation, the comment “Input“ stands for input parameter and the comment “Output“ for output parameter. These comments are no present at the structures, but on the lowest level at the parameters.

3.1 ft_cancel - Cancel asynchronous request

ft_cancel() cancels asynchronous requests which are in the course of being processed or which are waiting to be processed in the queue.

Syntax

```
#include <ftapi.h>

int ft_cancel ( const void *session,          /* input */
                long rid,                     /* input */
                struct ft_err *errorinfo,
                void *options );              /* input */
```

Parameters

session
: Session number in which the request is to be canceled.

rid
: ID of the request to be canceled.

  If the request to be canceled was issued in a different session, the current session must be assigned the same working directory as the one in which the request was issued.
  Furthermore, the program in which the asynchronous request is cancelled must be running under the same login name as the one in which the request was issued.

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
  The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options
: The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
  Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

0 No error

-1 Error. The error type is stored in *errorinfo*.

3.2 ft_close - Close session

You can use *ft_close()* to close a session opened with *ft_open()*. This function must be the last one called in a session. *ft_close()* releases resources that are no longer required. The session number is deleted and no subsequent reference to this session is possible.

Syntax

```
#include <ftapi.h>

int ft_close ( const void *session,          /* input */
               struct ft_err *errorinfo,
               void *options );              /* input */
```

Parameter

session
: Number of the session which is to be closed.

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options
: The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

0 No error

-1 Error. The error type is stored in *errorinfo*.

3.3 ft_credir - Create directory in remote system

ft_credir() creates a directory in the remote system.

Directory names must not exceed the length specified in the *maxrfnsize* field of the *ft_prop* structure, see section “ft_properties - Determine properties of the program interface” on page 30.

Syntax

```
#include <ftapi.h>

int ft_credir ( const struct ft_admission *admis,  /* input */
                const struct ft_crepar *par,       /* input */
                struct ft_err *errorinfo,
                void *options );                   /* input */
```

Parameters

admis

Specifications for the remote system (see section “ft_admission” on page 12).

par Specifications for the request which you declare with the structure *ft_crepar*:

```
struct ft_crepar
{
   int   creparvers;              /* input */
   char  *dn;                     /* input */
   char  *mgmtpasswd;             /* input */
   char  *fud;                    /* input */
   int   fudlen;                  /* input */
};
```

The fields of the *ft_crepar* structure have the following meanings:

creparvers

Version of the data structure.
The value `FT_CPARV1` must be entered for *creparvers*.

dn Name of the directory that is to be created in the remote system.

Absolute and relative path specifications are permitted. Relative path specifications refer to the user ID defined in the admission profile if the FTAC function is used, otherwise to the HOME directory.

mgmtpasswd
Password for the directory if it is password-protected.

fud Address of a data area for the so-called “Further Details“ which can indicate a more detailed cause of error if errors occur.
If NULL is specified then no more detailed error cause is output.
The *fud* parameter is only available if *creparvers* is set to the value `FT_CPARV1` and the *options* parameter is specified when *ft_credir* is called.

fudlen
Length of the data area for *fud*.
The *fudlen* parameter is only available if *creparvers* is set to the value `FT_CPARV1` and the *options* parameter is specified when *ft_credir* is called.

errorinfo
Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options
The specification of the *options* parameter is mandatory. The construction of the *ft_options* structure is described in section “ft_properties - Determine properties of the program interface” on page 30.

Return value

0 No error. The directory was created.

-1 Error. The directory was not created.
The error type is stored in *errorinfo*.

3.4 ft_delete - Delete file or directory in the remote system

You can use *ft_delete()* to delete a file or a directory in the remote system. File directories that are to be deleted must be empty.

In order to delete a **file**, the *filetype* parameter in the *par* structure must contain the value `FT_FILE`.

To delete a **directory**, the *filetype* parameter in the *par* structure must contain the value `FT_DIRECTORY`.

File names and directory names must not exceed the length specified in the *maxrfnsize* field of the *ft_prop* structure (see section “ft_properties - Determine properties of the program interface” on page 30).

Syntax

```
#include <ftapi.h>

int ft_delete ( const struct ft_admission *admis,  /* input */
                const struct ft_delpar *par,       /* input */
                struct ft_err *errorinfo,
                void *options );                   /* input */
```

Parameter

admis

Transfer admission for the remote system (see the section on “ft_admission” on page 12).

par Entries for delete request which you specify with the structure *ft_delpar*:

```
struct ft_delpar
{
   int   delparvers;              /* input */
   char  *fn;                     /* input */
   char  *mgmtpasswd;             /* input */
   enum  ft_filedir filetype;     /* input */
   char  *fud;                    /* input */
   int   fudlen;                  /* input */
};
```

The fields of the structure *ft_delpar* have the following meanings:

delparvers
: Version of the data structure.
delparvers must be supplied the value `FT_DPARV1` or `FT_DPARV2`.

fn
: Name of the file or directory to be deleted in the remote system.

  Absolute and relative path names are permissible. Relative path names refer to the login name specified in the FT profile, when the FTAC function is used, otherwise to the HOME directory, see page 10.

mgmtpasswd
: Password of the file/directory, if protected by a password.

filetype
: specifies what is to be deleted:

`FT_FILE`	File (Default value after initialization of the parameter list *ft_delpar* with binary 0)
`FT_DIRECTORY`	Directory (not for FTAM partners)

fud
: Address of a data area for the so-called “Further Details“ which can indicate a more detailed cause of error if errors occur.
If NULL is specified then no more detailed error cause is output.
The *fud* parameter is only available if *delparvers* is set to the value `FT_DPARV2` and the *options* parameter is specified when *ft_delete* is called.

fudlen
: Length of the data area for *fud*.
The *fudlen* parameter is only available if *delparvers* is set to the value `FT_DPARV2` and the *options* parameter is specified when *ft_delete* is called.

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options

The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

0 No error. The file or directory has been deleted.

-1 Error. The file or directory has not been deleted.
The error type is stored in *errorinfo*.

3.5 ft_open - Open session

You can use *ft_open()* to open a session. You can only transfer files asynchronously (function *ft_transfer()*) and manage asynchronous file transfer requests (functions *ft_reqlist()*, *ft_reqstat()*, *ft_cancel()* and *ft_reqterm()*) within a session.

ft_open() returns a session number which uniquely identifies the session. This session number must be specified for function calls issued in the same session.

You can open several session simultaneously in one program, provided that different working directories are assigned.

Syntax

```
#include <ftapi.h>

void *ft_open ( const char *workdir,          /* input */
                struct ft_err *errorinfo,
                void *options );              /* input */
```

Parameter

workdir

Name of the working directory assigned to the session.

Files containing management information are stored in this directory.

Please note that the password used to call the program interface must have authorization to store files in this directory.

errorinfo

Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options

The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

n Session number ($n \neq 0$).
This value must be specified for function calls issued in the same session.

NULL Error. The error type is stored in *errorinfo*.

3.6 ft_properties - Determine properties of the program interface

You can use *ft_properties()* to determine the version of the program interface of openFT, and version-specific system values. The values returned by the *ft_properties()* function allow to check whether your program has been created with the same or with a different version of the program interface.

Syntax

```
#include <ftapi.h>

int ft_properties ( struct ft_prop *prop,
                    struct ft_err *errorinfo );
```

Parameter

prop

Area in which the version of the openFT program interface used is stored, along with the valid system values. For this purpose, the structure *ft_prop* is used:

```
struct ft_prop
{
   int   ftpropvers;        /* input */
   int   ftvers;            /* output */
   long  optfunct;          /* output */
   int   maxlfnsize;        /* output */
   int   maxrfnsize;        /* output */
   int   maxsyssize;        /* output */
   int   maxadmissize;      /* output */
   int   maxaccsize;        /* output */
   int   maxpwdsize;        /* output */
   int   maxfpwdsize;       /* output */
   int   maxrecord;         /* output */
   int   maxacntsize;       /* output */
   int   maxlegalqsize;     /* output */
   int   maxcpwdsize;       /* output */
   int   maxlprocsize;      /* output */
   int   maxrprocsize;      /* output */
};
```

The fields of the structure *ft_prop* have the following meanings:

ftpropvers
: Version of the data structure.
ftpropvers must be supplied the value `FT_PROPV1`.

ftvers
: version of openFT
(e.g. for Version 8.0: `800`, for version 8.1: `810`, for version 10.0: `1000`)

optfunct
: (reserved for later use.)
optfunct has the value 0 in the current version.

maxlfnsize
: maximum length for the local file name

maxrfnsize
: maximum length for the file name in the remote system

maxsyssize
: maximum length for the name of the remote system

maxadmissize
: maximum length for the login name or transfer admission in the remote system

maxaccsize
: maximum length for the account number in the remote system

maxpwdsize
: maximum length for the password in the remote system

maxfpwdsize
: maximum length for the file password in the remote system

maxrecord
: maximum record length

maxacntsize
: maximum length for the account at the FTAM partner

maxlegalqsize
: maximum length for the copyright

maxcpwdsize
: maximum length for the password for creating a file in the remote system

maxlprocsize
: maximum overall length for local follow-up processing

maxrprocsize
: maximum overall length for remote follow-up processing

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

Return value

0 No error

-1 Error. The error type is stored in *errorinfo*.

3.7 ft_reqlist - Determine request not yet terminated

ft_reqlist() determines the request IDs of requests for asynchronous file transfer which have not yet been terminated with the function *ft_reqterm()*.
IF the *list* parameter has the value NULL or if the *listlen* parameter has the value 0, you only receive the number of requests that have not yet been completed when you use *ft_reqterm()*.

The request from all sessions assigned with the function *ft_open* to the same working directory that applies as the current session are listed.

Syntax

```
#include <ftapi.h>

int ft_reqlist ( const void *session,          /* input */
                 long *list,
                 int listlen,                  /* input */
                 struct ft_err *errorinfo,
                 void *options );              /* input */
```

Parameter

session
: Number of the session for which non-terminated asynchronous file transfer request is to be determined.

list
: Area in which the request IDs of non-terminated requests for asynchronous file transfer are stored. The length of this area (number of entries) must be stored in *listlen*.

  If *list* is NULL, only the number (not the request IDs) of non-terminated requests is determined.

listlen
: Number of entries in *list*.

  If *listlen* is 0, only the number (not the request IDs) of non-terminated requests is determined.

errorinfo

Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options

The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

n Number of entries found ($n \geq 0$).
If *n* is greater than *listlen*, the first *listlen* entries are stored in *list*.

-1 Error. The error type is stored in *errorinfo*.

3.8 ft_reqstat - Determine the status of a request

You can use *ft_reqstat()* to determine the status of an asynchronous file transfer request.

Syntax

```
#include <ftapi.h>

int ft_reqstat ( const void *session,          /* input */
                 long rid,                     /* input */
                 struct ft_status *stat,
                 struct ft_err *errorinfo,
                 void *options );              /* input */
```

Parameter

session
: Number of the session in which the status of the transfer request is to be determined.

rid
: ID of the request for which the status is to be determined.

  If the request was issued in a different session, the current session must be assigned the same working directory as the one in which the request was issued.

stat
: Area in which the status information is written. The structure *ft_status* is used:

```
#define STAT_FUD_LEN    65
#define STAT_FN_LEN    128

struct ft_status
{
   int   ftstatvers;         /* input */
   enum  ft_stat status;     /* output */
   char  fn[STAT_FN_         /* output */
   long  tid;                /* output */
   int   msg;                /* output */
   char  fud[STAT_FUD_LEN];  /* output */
};
```

ftstatvers

Version of the data structure.
ftstatvers must be supplied the value `FT_STATV1` or `FT_STATV2`.

status

Status of the request:

`FT_STATW`	The request is waiting for execution.
`FT_STATR`	The request is being run.
`FT_STATA`	The request was aborted.
`FT_STATT`	The request is terminated.

fn local file name terminating with '\0'. If the file name is longer than 128 characters, it is truncated.

tid Transfer ID

msg Message number of aborted or terminated requests (see the online help).

The *ft_apivers* field in the *ft_options* structure can be used to define the message number scheme that is to be used.

fud "Further Details" terminated with '\0' which can indicate a more detailed cause of error if errors occur.
The *fud* parameter is only available if *ftstatvers* is set to the value `FT_STATV2` and the *options* parameter is specified when *ft_reqstat* is called.

errorinfo

Area in which detailed information is stored if an error is encountered (see the section on "ft_err" on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options

The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section "ft_options" on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

0 No error

-1 Error. The error type is stored in *errorinfo*.

3.9 ft_reqterm - Terminate request

You can use t*ft_reqterm()* to terminate an asynchronous file transfer request. This is possible only if the request has the status „aborted“ or „completed“ . *ft_reqterm()* deletes the associated file containing the management information. Then the request ID is deleted and can no longer be addressed.

Syntax

```
#include <ftapi.h>

int ft_reqterm ( const void *session,          /* input */
                 long rid,                     /* input */
                 struct ft_err *errorinfo,
                 void *options );              /* input */
```

Parameter

session
: Number of the session in which the transfer request is to be terminated.

rid
: ID of the request to be terminated.

  If the request was issued in a different session, the current session must be assigned the same working directory as the one in which the request was issued.

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
  The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options
: The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
  Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

0 No error

-1 Error. The error type is stored in *errorinfo*.

3.10 ft_show - Determine attributes of a file or directory

You can use *ft_show()* to determine the attributes of an individual file or directory in the remote system. Use the function *ft_showdir()* to determine the attributes of several files.

File names and directory names must not exceed the length specified in the *maxrfnsize* field of the *ft_prop* structure (see section "ft_properties - Determine properties of the program interface" on page 30).

Syntax

```
#include <ftapi.h>

int ft_show ( const struct ft_admission *admis,     /* input */
              const struct ft_shwpar *par,          /* input */
              struct ft_fileinfo *info,
              struct ft_err *errorinfo,
              void *options );                      /* input */
```

Parameter

admis

Transfer admission for the remote system (see the section on "ft_admission" on page 12).

par Entries for the request which you specify with the structure *ft_shwpar*:

```
struct ft_shwpar
{
   int  shwparvers;      /* input */
   char *fn;             /* input */
   char *mgmtpasswd;     /* input */
   char *fud;            /* input */
   int  fudlen;          /* input */
};
```

The fields of the structure *ft_shwpar* have the following meanings:

shwparvers

Version of the data structure.
shwparvers must be supplied the value `FT_SPARV1` or `FT_SPARV2`.

fn Name of the file or directory for which the attributes are to be determined.

Absolute and relative path names are permissible. Relative path names refer to the login name specified in the FT profile, when the FTAC function is used, otherwise to the HOME directory, see page 10.

mgmtpasswd
Password of the file or directory if protected by a password.

fud Address of a data area for the so-called “Further Details“ which can indicate a more detailed cause of error if errors occur.
If NULL is specified then no more detailed error cause is output.
The *fud* parameter is only available if *shwparvers* is set to the value FT_SPARV2 and the *options* parameter is specified when *ft_show* is called.

fudlen
Length of the data area for *fud*.
The *fudlen* parameter is only available if *shwparvers* is set to the value FT_SPARV2 and the *options* parameter is specified when *ft_show* is called.

info Area in which the file attributes are written. The structure *ft_fileinfo* is used:

```
#define ACC_LEN       65
#define INFO_FN_LEN  257
#define LQ_LEN        81
#define USER_LEN      68

struct ft_fileinfo
{
   int   ftshowivers;                /* input */
   char  fn[INFO_FN_LEN];            /* output */
   enum  ft_ftype filetype;          /* output */
   enum  ft_charset charset;         /* output */
   enum  ft_rform recordform;        /* output */
   long  recsize;                    /* output */
   enum  ft_available availability;  /* output */
   int   acccess;                    /* output */
   char  accout[ACC_LEN];            /* output */
   long  size;                       /* output */
   long  maxsize;                    /* output */
   char  legalqual[LQ_LEN];          /* output */
   char  cre_user[USER_LEN];         /* output */
   long  cre_date;                   /* output */
```

```
    char  mod_user[USER_LEN];          /* output */
    long  mod_date;                    /* output */
    char  rea_user[USER_LEN];          /* output */
    long  rea_date;                    /* output */
    char  atm_user[USER_LEN];          /* output */
    long  atm_date;                    /* output */
    long  long fsize;                  /* output */
    long  long fmaxsize;               /* output */
};
```

The fields of the structure *ft_fileinfo* have the following meanings:

ftshowivers
: Version of the data structure.
ftshowivers must be supplied the value `FT_SHOWIV1` or `FT_SHOWIV2`.

fn
: file name or directory name

filetype
: file type:

`FT_TYPEUNKN`	File type unknown
`FT_BIN`	Binary file
`FT_DIR`	Directory
`FT_TXT`	Text file

charset
: Character set (only for text files):

`FT_NOSET`	Character set unknown
`FT_VISIBLE`	The file may contain characters from the G0 set of ISO646.
`FT_IA5`	The file may contain characters from the C0 set and the G0 set of ISO646.
`FT_GRAPHIC`	The file may contain characters from the ISO646 or from the G0 set of ISO8859-1 and the G1-Set of ISO8859-1.
`FT_GENERAL`	The file may contain characters from the C0 set of ISO646, from the G0 set of ISO646 or ISO8859-1 and from the G1 set of ISO8859-1.

recordform
: Record format:

FT_NOFORM	Record format unknown
FT_VARIABLE	variable-length records
FT_FIXED	fixed-length records
FT_UNDEF	undefined record length

recsize
: maximum record length or 0, if record length unknown

availability
: Availability of file:

FT_NOAVAIL	The availability is not specified.
FT_AVAILIMM	The file is immediately available.
FT_AVAILNIMM	The file is not immediately available.

access
: Access rights. The right is available if the bit is set.
The following bits are defined:

FT_ACCR	The file may be read.
FT_ACCI	File units may be added to the file.
FT_ACCP	The file may be overwritten.
FT_ACCX	The file may be extended, i.e. data can be added to the file.
FT_ACCE	File units may be deleted from the file.
FT_ACCA	File attributes may be read.
FT_ACCC	File attributes may be modified.
FT_ACCD	The file may be deleted.

account
: account number used to charge costs in the remote system

size
: current file size in bytes, or -1 if file size unknown. In systems in which variables of type `long` have a size of 32 bits, the value for the file size is truncated if it no longer fits in the field. The complete value for the file size can be found in the *fsize* field.

maxsize
: permissible file size, or -1 if file size unknown. In systems in which variables of type `long` have a size of 32 bits, the value for the file size is truncated if it no longer fits in the field. The complete value for the file size can be found in the *fmaxsize* field.

legalqual
: legal qualification

cre_user
: user who created the file

cre_date
: time at which file was created, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

mod_user
: user who last modified the file contents

mod_date
: time at which the file contents were last modified, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

rea_user
: user who read the file last

rea_date
: time at which the file was last read, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

atm_user
: User who last modified the file attributes

atm_date
: time at which the file attributes were last modified, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

fsize

Current file size in bytes or -1 if the file size is unknown. The *fsize* parameter is only available if *ftshowivers* is set to the value FT_SHOWIV2 and the *options* parameter is specified when *ft_show* is called.

fmaxsize

Current file size in bytes or -1 if the permitted file size is unknown. The *fmaxsize* parameter is only available if *ftshowivers* is set to the value FT_SHOWIV2 and the *options* parameter is specified when *ft_show* is called.

errorinfo

Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options

The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

0 No error.

-1 Error. No information supplied about the file.
The error type is stored in *errorinfo*.

3.11 ft_showdir - Determine the attributes of all file in a directory

You can use *ft_showdir()* to determine the attributes of all files in a directory in the remote system. Please note that the attributes of all file are determined and that you cannot select files within a directory.

Use the function *ft_show()* to determine the attributes of an individual file/directory.

Directory names must not exceed the length specified in the *maxrfnsize* field of the *ft_prop* structure (see section “ft_properties - Determine properties of the program interface” on page 30).

Syntax

```
#include <ftapi.h>

long ft_showdir ( const struct ft_admission *admis,/* input */
                  const struct ft_shwpar *par,     /* input */
                  struct ft_fileinfo *buf,
                  int bufsize,                     /* input */
                  struct ft_err *errorinfo,
                  void *options );                 /* input */
```

Parameter

admis

Transfer admission of the remote system (see the section on “ft_admission” on page 12).

par

Parameters for the request, which you specify with the structure *ft_shwpar*:

```
struct ft_shwpar
{
   int  shwparvers;      /* input */
   char *fn;             /* input */
   char *mgmtpasswd;     /* input */
   char *fud;            /* input */
   int  fudlen;          /* input */
};
```

The fields of the structure *ft_shwpar* have the following meanings:

shwparvers
: Version of the data structure.
shwparvers must be supplied the value `FT_SPARV1` or `FT_SPARV2`.

fn
: Name of the directory for which the attributes are to be determined.

  Absolute and relative path names are permissible. Relative path names refer to the login name specified in the FT profile, when the FTAC function is used, otherwise to the HOME directory, see page 10.

mgmtpasswd
: Password of the directory if it is password-protected.

fud
: Address of a data area for the so-called “Further Details“ which can indicate a more detailed cause of error if errors occur.
If NULL is specified then no more detailed error cause is output.
The *fud* parameter is only available if *shwparvers* is set to the value `FT_SPARV2` and the *options* parameter is specified when *ft_showdir* is called.

fudlen
: Length of the data area for *fud*.
The *fudlen* parameter is only available if *shwparvers* is set to the value `FT_SPARV2` and the *options* parameter is specified when *ft_showdir* is called.

buf
: Area in which the file attributes are written. The area comprises elements with the structure *ft_fileinfo*:

```
#define ACC_LEN       65
#define INFO_FN_LEN  257
#define LQ_LEN        81
#define USER_LEN      68

struct ft_fileinfo
{
   int   ftshowivers;                  /* input  */
   char  fn[INFO_FN_LEN];              /* output */
   enum  ft_ftype filetype;            /* output */
   enum  ft_charset charset;           /* output */
   enum  ft_rform recordform;          /* output */
   long  recsize;                      /* output */
   enum  ft_available availability;    /* output */
```

```
    int   acccess;                    /* output */
    char  accout[ACC_LEN];            /* output */
    long  size;                       /* output */
    long  maxsize;                    /* output */
    char  legalqual[LQ_LEN];          /* output */
    char  cre_user[USER_LEN];         /* output */
    long  cre_date;                   /* output */
    char  mod_user[USER_LEN];         /* output */
    long  mod_date;                   /* output */
    char  rea_user[USER_LEN];         /* output */
    long  rea_date;                   /* output */
    char  atm_user[USER_LEN];         /* output */
    long  atm_date;                   /* output */
    long  long fsize;                 /* output */
    long  long fmaxsize;              /* output */
};
```

The fields of the structure *ft_fileinfo* have the following meanings:

ftshowivers
: Version of the data structure.
ftshowivers must be supplied the value `FT_SHOWIV1` or `FT_SHOWIV2`. *ftshowivers* need only be set in the first passed data structure.

fn
: file name or directory name

filetype
: file type:

`FT_TYPEUNKN`	File type unknown
`FT_BIN`	Binary file
`FT_DIR`	Directory
`FT_TXT`	Text file

charset
: Character set (only for text files):

FT_NOSET	Character set unknown
FT_VISIBLE	The file may contain characters from the G0 set of ISO646.
FT_IA5	The file may contain characters from the C0 set and the G0 set of ISO646.
FT_GRAPHIC	The file may contain characters from the ISO646 or from the G0 set of ISO8859-1 and the G1-Set of ISO8859-1.
FT_GENERAL	The file may contain characters from the C0 set of ISO646, from the G0 set of ISO646 or ISO8859-1 and from the G1 set of ISO8859-1.

recordform
: Record format:

FT_NOFORM	Record format unknown
FT_VARIABLE	variable-length records
FT_FIXED	fixed-length records
FT_UNDEF	undefined record length

recsize
: maximum record length or 0, if the maximum record length is unknown

availability
: Availability of file:

FT_NOAVAIL	The availability is not specified.
FT_AVAILIMM	The file is immediately available.
FT_AVAILNIMM	The file is not immediately available.

access
: Access rights. The right is available if the bit is set. The following bits are defined:

`FT_ACCR`	The file may be read.
`FT_ACCI`	File units may be added to the file.
`FT_ACCP`	The file may be overwritten.
`FT_ACCX`	The file may be extended, i.e. data can be added to the file.
`FT_ACCE`	File units may be deleted from the file.
`FT_ACCA`	File attributes may be read.
`FT_ACCC`	File attributes may be modified.
`FT_ACCD`	The file may be deleted.

account
: account number used to charge costs in the remote system

size
: current file size in bytes, or -1 if file size unknown. In systems in which variables of type `long` have a size of 32 bits, the value for the file size is truncated if it no longer fits in the field. The complete value for the file size can be found in the *fsize* field.

maxsize
: permissible file size, or -1 if file size unknown. In systems in which variables of type `long` have a size of 32 bits, the value for the file size is truncated if it no longer fits in the field. The complete value for the file size can be found in the *fmaxsize* field.

legalqual
: legal qualification

cre_user
: user who created the file

cre_date
: time at which file was created, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

mod_user
: user who last modified the file contents

mod_date
: time at which the file contents were last modified, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

rea_user
: user who read the file last

rea_date
: time at which the file was last read, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

atm_user
: file user who last modified the file attributes

atm_date
: time at which the file attributes were last modified, or 0 if time unknown.
The time is specified in internal format (seconds since 1.1.1970 00:00:00).

fsize
: Current file size in bytes or -1 if the file size is unknown. The *fsize* parameter is only available if *ftshowivers* is set to the value `FT_SHOWIV2` and the *options* parameter is specified when *ft_showdir* is called.

fmaxsize
: Current file size in bytes or -1 if the permitted file size is unknown. The *fmaxsize* parameter is only available if *ftshowivers* is set to the value `FT_SHOWIV2` and the *options* parameter is specified when *ft_showdir* is called.

bufsize
: size of *buf*, i.e. number of elements with the structure *ft_fileinfo*

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options

The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section "ft_options" on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

n Number of files found in the remote directory ($n \geq 0$).
If *n* is greater than *bufsize*, the first *bufsize* entries are stored in *buf*.
If *buf* has the value NULL, the function call behaves as though *bufsize* had the value 0.

-1 Error. No information was returned for the directory.
The error type is stored in *errorinfo*.

3.12 ft_transfer - Transfer file

ft_transfer() sends a file to the remote system or fetches a file from the remote system.

In order to transfer files **synchronously**, the parameter *synchron* must contain the value `FT_SYNC`.

In order to transfer files **asynchronously**, the parameter *synchron* must contain the value `FT_ASYNC`.
For asynchronous file transfer, the function *ft_transfer()* returns a request ID which you must specify when you refer to this request.

File names must not exceed the length specified in the *maxlfnsize* or *maxrfnsize* fields of the *ft_prop* structure (see section “ft_properties - Determine properties of the program interface” on page 30.

Syntax

```
#include <ftapi.h>

long ft_transfer ( const void *session,            /* input */
                   const struct ft_admission *admis,
                                                   /* input */
                   const struct ft_transpar *par,  /* input */
                   struct ft_err *errorinfo,
                   void *options );                /* input */
```

Parameter

session
: For asynchronous transfer:
Number of the session in which the transfer request is to be performed.

  For synchronous transfer:
session must have the value NULL.

admis
: Transfer admission for the remote system (see the section on “ft_admission” on page 12).

par

Entries for the request which you specify with the structure *ft_transpar*:

```
struct ft_transpar
{
   int    ftparvers;                  /* input */
   enum   ft_direction direction;     /* input */
   enum   ft_sync synchron;           /* input */
   char   *locfn;                     /* input */
   char   *remfn;                     /* input */
   enum   ft_filetype filetype;       /* input */
   enum   ft_writemode writemode;     /* input */
   enum   ft_compress compress;       /* input */
   char   *filepasswd;                /* input */
   char   *locsuccproc;               /* input */
   char   *locfailproc;               /* input */
   char   *remsuccproc;               /* input */
   char   *remfailproc;               /* input */
   long   maxrecsize;                 /* input */
   long   cantime;                    /* input */
   long   starttime;                  /* input */
   enum   ft_prio priority;           /* input */
   enum   ft_transpar transparent;    /* input */
   enum   ft_encrypt encryption;      /* input */
   struct ft_transftam *ftamext;      /* input */
   char   *locccsn;                   /* input */
   char   *remccsn;                   /* input */
   enum   ft_tabexp tabexp;           /* input */
   char   *fud;                       /* input */
   int    fudlen;                     /* input */
   enum   ft_rform rform;             /* input */
};
```

Default values apply to certain parameters (see below) if you initialize this parameter list with the entries for the file transfer request with binary 0.

To initialize the parameter list with binary 0, use the command:

```
memset (transpar, '\0', sizeof(struct ft_transpar));
```

The fields of the structure *ft_transpar* have the following meanings:

ftparvers
: Version of the data structure.
ftparvers must be supplied the value `FT_TPARV1` or `FT_TPARV2`.

direction
: Direction of file transfer:

`FT_SEND`	Send file to the remote system
`FT_RECEIVE`	Fetch file from remote system. You cannot use wildcards when fetching a file.

synchron
: specifies how the file is to be transferred:

`FT_ASYNC`	asynchronous transfer (default value after initialization with binary 0)
`FT_SYNC`	synchronous transfer

locfn
: File name in local system or preprocessing/postprocessing command.

  In the case of local file names, it is now also possible to specify a preprocessing (when sending data) or postprocessing (when receiving data) command.

  Absolute and relative path names are permissible. Relative path names refer to the directory in which the program is started.

remfn
: File name in the remote system or preprocessing/postprocessing command.

  In the case of remote file names, it is now also possible to specify a preprocessing (when sending data) or postprocessing (when receiving data) command.

  Absolute and relative path names are permissible. Relative path names refer to the login name specified in the FT profile when the FTAC function is used, otherwise to the HOME directory, see page 10. As in the command ncopy, a preprocessing command may be specified next to it instead of a file name by using a preceding “or” (|) character (see also the command description of ncopy).

filetype

File type in local system:

FT_NOTYPE	No specification of file type. The default values apply (see online help, *ft* command). (default value after installation with binary 0)
FT_TEXT	The file contains text with variable record lengths. Records are delimited in Windows by CRLF (X'0D0A') and in Unix systems by the linefeed character \n. Maximum record length: 32 767 bytes
FT_USER	The file contains binary data with variable record length structured by the user. Each record starts with two bytes which indicate the length of the record. Maximum record length: 32 767 bytes
FT_BINARY	The file contains an unstructured sequence of binary data

writemode

specifies whether a new destination file is to be created or extended:

FT_NOMODE	No syntax is specified. The default values apply (see online help, *ft* command). (default value after initialization with binary 0)
FT_OVERWR	An existing destination file is overwritten. If the destination does not already exist, a new one is created.
FT_EXTEND	The file transferred is added at the end of the existing destination file.If the destination does not already exist, a new one is created.
FT_NEW	A new destination file is created and written. IF the destination file already exists, the request is rejected.

compress
: specifies whether data compression is to be used in transfer:

  `FT_NOCOMPR` No compression (default value after initialization with binary 0)

  `FT_COMPRESS` Several identical characters in succession are transferred in compressed form (byte compression).

  `FT_COMPRESSZIP`
  Zip compression. In the case of connections to partners which do not support this compression, operation automatically switches to byte compression or no compression. Zip compression is only available if *ftparvers* is set to the value `FT_TPARV2` and the *options* parameter is specified when *ft_transfer* is called.

filepasswd
: Password of the file in the remote system, if protected by password.

locsuccproc
: Command executed in the local system following successful asynchronous file transfer.

  locsuccproc must not be specified for synchronous file transfer requests.

  Variables can be specified within a command or sequence of commands for follow-up processing. Further information is given in the section “Commands for the follow-up processing” on page 59.

locfailproc
: command executed in the local system when an asynchronous file transfer is aborted due an error.

  With synchronous file transfer request, *locfailproc* must not be specified.

  Variables can be specified within a command or sequence of commands for follow-up processing. Further information is given below in the section “Commands for the follow-up processing” on page 59.

remsuccproc
: Command executed in the remote system following successful asynchronous file transfer.

  Several partner systems(e.g. openFT for BS2000/OSD) even support sequence of commands. Following successful transfer, these commands are executed in the remote system under the specified login.

  Variables can be specified within a command or sequence of commands for follow-up processing. Further information is given below in the section “Commands for the follow-up processing” on page 59".

remfailproc
: Command executed in the remote system following unsuccessful asynchronous file transfer.

  Several partner systems(e.g. openFT for BS2000/OSD) even support sequence of commands. These commands are executed in the remote system under the specified login when file transfer that has been started is aborted due to an error.

  Variables can be specified within a command or sequence of commands for follow-up processing. Further information is given below in the section "Commands for follow-up processing".

Commands for the follow-up processing

– The total number of entries for local follow-up processing, i.e. for *locsuccproc* and *locfailproc*, may not exceed 1000 characters.

– The total number of entries for remote follow-up processing, i.e. for *remssuccproc* and *remfailproc*, may not exceed 1000 characters, but may be lower if a FT version < V6 is used in the remote system.

– When starting follow-up processing in the local system, the variables are substituted with the values supplied by the *ft_transfer()* function.

  The variable %FILENAME is provided for the file name, %PARTNER for the partner name, %RESULT for the result of the request and %RID for the request ID.

  %RID is only allowed for local follow-up processing.

  After follow-up processing is started, the variables in the particular system are replaced and the commands in the follow-up processing are executed. The following variables are permitted:

  – %FILENAME
    by the file name as specified for the corresponding system in the request

  – %PARTNER
    for local follow-up processing the partner name specified in the call is used.
    For follow-up processing in the remote system, %PARTNER is substituted by the name of the initiator system (with the name as known in the partner system).

  – %RESULT
    the message number of the request refers to the relevant system. If, for example, a send request is successfully executed, the value of %RESULT in the local system is 82 for „Send request successful“, while the value in the remote system is 81 for „Receive request successful.“

– %RID
by the request ID of the request in the local system (only local follow-up processing)

If the partners partner is an openFT for BS2000/OSD system, you may also use the variables %ELEMNAME, %ELEMVERS and %ELEMTYPE.

– During follow-up processing in the local Windows system, only the system environment variables are available.

– Follow-up processing in the local Unix system and follow-up processing in a remote Unix does not involve execution of the sequence of commands stored in the *.profile* file. Only the default values of the $HOME, $LOGNAME, $PATH, and $USER shell variables are available, as well as the values of the $LANG and $TZ variables set by *root*.

– When specifying BS2000 commands, remember to insert a slash (/) at the beginning of the command

– With requests for FTAM and FTP partners, only the “local follow-up processing” function is available. If FTAC is used in the remote system, this restriction can be avoided by creating an FT profile in the remote system and defining follow-up processing for it.

maxrecsize
maximum permissible record length for files of type „text file“ and „structured binary file“. Thus, it is also possible to transfer and store records which are larger than the default value. However, you must observe that not all record lengths can be preprocessed in every partner system.

The maximum value is 32676 bytes.

If you have selected the file type „binary“, *maxrecsize* id the value for all records in the send file.

With FTAM partners, specification of the maximum record length is only effective when the file type `FT_TEXT`, `FT_USER` or `FT_BINARY` is specified for *filetype*.

cantime

Time at which a file transfer request is to be canceled. This time must be specified in internal format (seconds since 1.1.1970 00:00:00).

The value 0 means that no time cancellation is performed.

With synchronous requests, *cantime* is ignored.

starttime

specifies the earliest time at which file transfer is to be started. This time must be specified in internal format (seconds since 1.1.1970 00:00:00).

The value 0 means that the file transfer is started as soon as possible.

For synchronous requests, *starttime* is ignored.

priority

specifies the priority of the request:

`FT_PRIONORM`	Normal priority (default value after initialization with binary 0)
`FT_PRIOLOW`	Low priority (is ignored for synchronous requests)

transparent

specifies whether the file transfer is to be transparent:

`FT_NOTRANSPAR`	normal transfer (default value after initialization with binary 0)
`FT_TRANSPARENT`	Transparent file transfer

encryption

specifies whether the user data are to be encrypted or whether a data integrity check is to be performed:

`FT_NOENCRYPT`	User data are not encrypted and no data integrity check is performed (default value after initialization with binary 0)
`FT_ENCRYPT`	User data are encrypted and data integrity is checked automatically. openFT-CR must be installed to enable this.
`FT_ONLYDICHECK`	A data integrity check is performed for the transferred file contents. The data integrity check is only available if *ftparvers* is set to the value `FT_TPARV2` and the *options* parameter is specified when *ft_transfer* is called.

ftamext

FTAM-specific parameter made known with the structure *ft_transftam* (see also the commands *ft* and *ncopy*, options *-av*, *-ac*, *-am*, *-lq* and *-cp*):

```
struct ft_transftam
{
   enum ft_available available;     /* input */
   char  *account;                  /* input */
   int   accessmode;                /* input */
   char  *legalq;                   /* input */
   char  *crpasswd;                 /* input */
};
```

The fields of the structure *ft_transftam* have the following meanings:

available
: specifies the availability of the destination file:

FT_NOAVAIL	No specification of availability (default value after initialization with binary 0)
FT_AVAILIMM	The destination file contains the attribute „immediately available“.
FT_AVAILNIMM	The destination file contains the attribute „not immediately available“.

account
: account number for FTAM partners

accessmode
: specifies the access rights for the destination file. The access rights are created by logical ORing of the individual rights:

FT_ACCR	The file may be read.
FT_ACCI	File units may be inserted into the file.
FT_ACCP	The file may be overwritten.
FT_ACCX	The file may be extended, i.e. data can be appended to the file.
FT_ACCE	File units may be deleted from the file.
FT_ACCA	File attributes may be read.
FT_ACCC	File attributes may be modified.
FT_ACCD	The file may be deleted.

legalq
: stipulates the copyright for the destination file.

crpasswd
: Password required to create a file in the remote system.

locccsn

Specifies the name of the coding (CCS name) used to read or write the local file. *CCS-name* must be known in the local system.

If no coding is specified then the default coding value set for openFT via the operating parameters is used.

Support for “Coded Character Sets“ (CCS) is only available if *ftparvers* is set to the value `FT_TPARV2` and the *options* parameter is set when *ft_transfer* is called.

remccsn

Specifies the name of the coding (CCS name) used to read or write the remote file. *CCS-name* must be known in the remote system. If no coding is specified then character set defined via XHCS (BS2000/OSD) or via the openFT operating parameters (other platforms) is used for coding.

Support for “Coded Character Sets“ (CCS) is only supported for the openFT protocol and for partners with openFT V10.0 or higher and is only available if *ftparvers* is set to the value `FT_TPARV2` and the *options* parameter is set when *ft_transfer* is called.

tabexp

In the case of an outbound send request, specifies whether tabulator expansion and the conversion of blank lines into lines with a character for non-FTAM partners are to be performed. Tabulator expansion is only available if *ftparvers* has the value `FT_TPARV2` and the *options* parameter is specified when *ft_transfer* is called.

`FT_TABAUTO`	Tabulator expansion and the conversion of blank lines are activated when a file is sent to a BS2000, OS/390 or z/OS system (default value after initialization with binary 0).
`FT_TABON`	Tabulator expansion and the conversion of blank lines are activated.
`FT_TABOFF`	Tabulator expansion and the conversion of blank lines are deactivated.

fud
: Address of a data area for the so-called “Further Details““ which can indicate a more detailed cause of error if errors occur.
If NULL is specified then no more detailed error cause is output.
The *fud* parameter is only available if *ftparvers* is set to the value FT_TPARV2 and the *options* parameter is specified when *ft_transfer* is called.

fudlen
: Length of the data area for *fud*.
The *fudlen* parameter is only available if *ftparvers* is set to the value FT_TPARV2 and the *options* parameter is specified when *ft_transfer* is called.

rform
: Specifies the record format of the file that is to be transferred. The *rform* parameter is only available if *ftparvers* is set to the value FT_TPARV2 and if the *options* parameter is specified when *ft_transfer* is specified.

FT_NOFORM	The record format of the file that is to be transferred is unknown (default value after initialization with binary 0).
FT_VARIABLE	The file to be transferred contains records with a variable record length.
FT_FIXED	The file to be transferred contains records with a standard, fixed record length.
FT_UNDEF	The file to be transferred contains records with an undefined record length.

errorinfo
: Area in which detailed information is stored if an error is encountered (see the section on “ft_err” on page 15).
The specification of this parameter is optional. If you do not require any more precise error information then you can specify the value NULL for *errorinfo*.

options
: The specification of the *options* parameter is optional. If the value NULL is specified then message activity at the program interface is compatible with that of the previous program interface.
Alternatively, it is possible to specify the *ft_options* structure (see the section “ft_options” on page 16) to activate the new openFT message number scheme and the extensions to the function.

Return value

n	For successful asynchronous requests: request ID ($n \neq 0$)
1	For successful synchronous requests
0	Error. File transfer was not initiated. The error type is stored in *errorinfo*.

4 Error codes

Error codes entered in the structure *ft_err* (see the section on “ft_err” on page 15), are made up of the following fields:

– main (error class)
– detail (error)
– additional (additional error information)

The error messages are sorted according to error classes in the list below. The following error classes are used:

FTEM_INT	Internal error
FTEM_PAR	Parameter error
FTEM_LOCERR	Sequence error in the local system
FTEM_CONNERR	Sequence error in the connection to the partner
FTEM_REMERR	Sequence error in the remote system

Table 1:

In the list below, error messages classified as parameter errors are assigned to the function calls in cases where the errors are not general.

4.1 Internal errors

Error class	Error	Additional error information	Meaning
FTEM_INT	FTED_MEM	0	Error on request for memory
FTEM_INT	FTED_CRFILE	0	Error on creation of the file
FTEM_INT	FTED_INIT	0	The server cannot be initialized
FTEM_INT	FTED_SIGNAL	*signal*	Command interrupted by *signal* . *signal* designates the signal that caused the interruption.
FTEM_INT	*function*	*errno*	Error on system call. *function* designates the errored system call: FTED_FORK fork FTED_OPEN open FTED_OPENDIR opendir FTED_PIPE pipe FTED_READ read FTED_RMFILE rmfile FTED_STAT stat FTED_SYSTEM system FTED_WRITE write *errno* is the value of the errno variable set by the errored system call. If not all bytes could be written, *errno* has the value −1.
FTEM_INT	FTED_INTERNAL	*reason*	Other internal error. *reason* designates the cause of the error, if known: FTEA_FN function not supported by server FTEA_VERS Version of file structure not supported by server

4.2 Parameter errors

General errors

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_INVSESS	0	The session number is invalid.
FTEM_PAR	FTED_LEN	*parameter*	*parameter* designates the parameter which is too long: FTEA_REMACC remaccount FTEA_REMADM remadmis FTEA_REMPED rempasswd FTEA_REMSYS remsys
FTEM_PAR	FTED_MAND	FTEA_REMSYS	The remote system was not specified.
FTEM_PAR	FTED_VALUE	FTEA_APIVERS	The API version specified in *ft_options* is invalid.
FTEM_PAR	FTED_VALUE	FTEA_REMSYS	The remote system is unknown.
FTEM_PAR	FTED_VALUE	FTEA_RID	The request ID (rid) is invalid.
FTEM_PAR	FTED_VERS	0	The version of the file structure (parameter list or output range) is invalid.

Error for ft_cancel

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_TERM	0	The request is already terminated.

Errors for ft_credir

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_LEN	*parameter*	*parameter* designates the parameter, which is too long: FTEA_FPWD mgmtpasswd FTEA_REMFN dn
FTEM_PAR	FTED_MAND	0	The parameter list *par* was not specified.
FTEM_PAR	FTED_REMOTE	FTEA_NOACCESS	No authorization to create in remote system
FTEM_PAR	FTED_REMOTE	FTEA_EXIST	Directory already exists in remote system.
FTEM_PAR	FTED_VALUE	*parameter*	*parameter* designates the parameter which is invalid: 0 unknown parameter/parameters are incompatible FTEA_FPWD mgmtpasswd FTEA_REMACC remaccount FTEA_REMADM remadmis FTEA_REMFN remfn FTEA_REMPWD rempasswd FTEA_REMSYS remsys

Error for ft_delete

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_LEN	*parameter*	*parameter* designates the parameter which is too long: FTEA_FPWD mgmtpasswd FTEA_REMFN fn
FTEM_PAR	FTED_MAND	0	The parameter list *par* was not specified.
FTEM_PAR	FTED_REMOTE	FTEA_NOACCESS	No authorization to delete in remote system
FTEM_PAR	FTED_REMOTE	FTEA_NOTEMPTY	The directory in the remote system is not empty.
FTEM_PAR	FTED_REMOTE	FTEA_NOTEXIST	File/directory does not exist in remote system.
FTEM_PAR	FTED_VALUE	*parameter*	*parameter* designates the parameter which is invalid: 0 unknown parameter/ parameters are incompatible. FTEA_FPWD mgmtpasswd FTEA_FTYPE filetype FTEA_REMACC remaccount FTEA_REMADM remadmis FTEA_REMFN remfn FTEA_REMPWD rempasswd FTEA_REMSYS remsys

Error for ft_open

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_DIRAC	*errno*	*errno* designates the value of the errno variable set by the stat() call. The errno variable has the value 0, if no write access is assigned to the directory.
FTEM_PAR	FTED_LEN	0	The name of the working directory (workdir) is too long.
FTEM_PAR	FTED_MAND	0	The name of the working directory (workdir) was not specified.
FTEM_PAR	FTED_NODIR	0	The specified name (workdir) is not a directory.
FTEM_PAR	FTED_OPEN	0	A session in a program has already been assigned this working directory (workdir).
FTEM_PAR	FTED_VALUE	0	The name for the working directory (workdir) is invalid.

Error for ft_reqstat

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_MAND	0	The output range *stat* was not specified.

Error on ft_reqterm

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_NOTERM	0	The request is still active.

Error for ft_show and ft_showdir

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_LEN	*parameter*	*parameter* designates the parameter which is too long: FTEA_FPWD mgmtpasswd FTEA_REMFN fn
FTEM_PAR	FTED_MAND	0	The parameter list *par* was not specified. / The output range *info* was not specified (only for *ft_show()*).
FTEM_PAR	FTED_REMOTE	FTEA_NOACCESS	No authorization to read attributes in the remote system
FTEM_PAR	FTED_REMOTE	FTEA_NOTEXIST	File/directory does not exist in the remote system.
FTEM_PAR	FTED_VALUE	*parameter*	*parameter* designates the parameter which is invalid: 0 unknown parameter/ parameters are incompatible. FTEA_FPWD mgmtpasswd FTEA_REMACC remaccount FTEA_REMADM remadmis FTEA_REMFN remfn FTEA_REMPWD rempasswd FTEA_REMSYS remsys

Error for ft_transfer

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_LEN	*parameter*	*parameter* designates the parameter which is too long: FTEA_ACCOUNT ftamext -> account FTEA_CRPWD ftamext -> crpasswd FTEA_FPWD filepasswd FTEA_LEGALQ ftamext -> legalq FTEA_LOCCCSN locccsn FTEA_LOCFN locfn FTEA_LOCPR Sum of the lengths of locsuccproc and locfailproc FTEA_REMCCSN remccsn FTEA_REMFN remfn FTEA_REMPR Sum of the lengths of remsuccproc and remfailproc
FTEM_PAR	FTED_MAND	*parameter*	*parameter* designates the parameter which was not specified: 0 The parameter list *par* was not specified. FTEA_LOCFN locfn

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_VALUE	*parameter*	*parameter* designates the parameter which is invalid: FTEA_ACCESS ftamext -> accessmode FTEA_AVAIL ftamext -> available FTEA_CANTIME cantime FTEA_COMPR compress FTEA_DIR direction FTEA_ENCRYPT encryption FTEA_FTYPE file type FTEA_MAXREC maxrecsize FTEA_PRIO priority FTEA_REMADM remadm. The user ID/ transfer admission in the remote system is invalid. FTEA_REMFN remfn. The specified file does not exist/ no access permitted. FTEA_REMSYS remsys. The specified remote system is unknown. FTEA_SYNC synchronous FTEA_RFORM rform FTEA_STARTTIME start time FTEA_TABEXP tabexp FTEA_TRANSP transparent FTEA_WMODE write mode

4.3 Sequence errors

General errors

Error class	Error	Additional error information	Meaning
FTEM_PAR	FTED_FTMSG	*code*	*code* designates the message number of the corresponding command (see the online help). The associated message text can also be determined using the *fthelp code* command.

Error in the local system

Error class	Error	Additional error information	Meaning
FTEM_LOCERR	FTED_EXIST	0	The local file already exists.
FTEM_LOCERR	FTED_FTAC	0	The request was rejected by the local FTAC.
FTEM_LOCERR	FTED_INCONS	0	The local file is inconsistent.
FTEM_LOCERR	FTED_MEM	0	The local file has no memory.
FTEM_LOCERR	FTED_NOACCESS	0	The local file cannot be accessed.
FTEM_LOCERR	FTED_NOCREAT	0	The local file cannot be created.
FTEM_LOCERR	FTED_NOTEXIST	0	The local file cannot be found.

Error in connection to the remote system

Error class	Error	Additional error information	Meaning
FTEM_CONNERR	FTED_NOCONN	0	No free transport connection
FTEM_CONNERR	FTED_NOTAVAIL	0	The remote system is not available.
FTEM_CONNERR	FTED_UNKNOWN	0	The remote system is unknown.

Error in remote system

Error class	Error	Additional error information	Meaning
FTEM_REMERR	FTED_EXIST	0	The remote file does already exists.
FTEM_REMERR	FTED_INCONS	0	The remote file is inconsistent.
FTEM_REMERR	FTED_MEM	0	The remote file has no memory.
FTEM_REMERR	FTED_NOACCESS	0	The remote file cannot be accessed.
FTEM_REMERR	FTED_NOCREAT	0	The remote file cannot be created.
FTEM_REMERR	FTED_NOTEXIST	0	The remote file cannot be found.
FTEM_REMERR	FTED_REMADM	0	The remote transfer admission is invalid.

5 Sample programs

The following sample programs show you the various options for using the program interface. The source codes of these programs can be found in the following subdirectory of the openFT installation directory:

– Windows: *openFT\samples\ftapi*
– Unix systems: */opt/openFT/samples*

Sample 1: Asynchronous file transfer of a file

The following program *sample1* is called in this way:

`sample1` *file1 file2*

Name and transfer admission to the remote system are then queried in a dialog. For the program to run, the following directory must exist:

– Windows: the *ft* subdirectory in the variable *TMP*.
– Unix systems: the working directory `$HOME/ft`

The program transfers the file *file1* asynchronously from the local system to remote system and stored it there under the name *file2* in the HOME directory of the user or under the login name specified in the T profile. The precondition for this is that the file to be sent, *file1*, is located in the same directory as the one in which the program is called. If a SIGINT signal is created by the user (e.g by entering CTRL+C under Windows), file transfer is aborted, provided that it has not yet been completed.

The program is structured as follows:

Since the file is to be transferred asynchronously, a session is first opened with the function *ft_open()*, where `%TMP%\ft` (under Windows) resp. `$HOME/ft` (under Unix systems) is permanently assigned as working directory. *ft_open()* returns a session number, which identifies the session and must be specified with further function calls. The asynchronous file transfer is initiated with the *ft_transfer()* function, which returns the request ID for the request. From now on, the program queries whether a SIGINT signal was created by the user. If so, the request is aborted with the *ft_cancel()* function. As long as file transfer has not been terminated or aborted, the *ft_reqstat()* queries the status of the file transfer request. If file transfer is completed or has been aborted, the request is marked as terminated with the *ft_reqterm()* function and the session closed with the *ft_close()* function.

Windows

```
/*************************************************************/
/* sample1.c  File transfer with standard parameters         */
/*************************************************************/
/*                                                           */
/* Component Name  : sample1                                 */
/*                                                           */
/* Subcomponent of : SAMPLES                                 */
/*                                                           */
/* Systemdependency: WINDOWS                                 */
/*                                                           */
/* Copyright (c) Fujitsu Technology Solutions 2009           */
/*               All Rights Reserved                         */
/*                                                           */
/*************************************************************/
static const char Sccsid[] = "@(#)sample1.c   2.33 2009/03/25";
/*************************************************************

  Program call: sample1 <local file> <remote file>

 *************************************************************/
/*                     Include Files                         */
/*************************************************************/
#include <windows.h>           /* Sleep()                    */
#include <stdio.h>             /* printf()                   */
#include <stdlib.h>            /* exit()                     */
#include <string.h>            /* strcat(), strcpy()         */
#include <signal.h>            /* signal()                   */
#include <sys/stat.h>          /* stat()                     */
#include <sys/types.h>         /* stat()                     */
#include <ftapi.h>
/*FF*/

/*************************************************************/
/*                     Global Variables                      */
/*************************************************************/
void *session_no;    /* Session identification               */
long  req_id;        /* Request ID                           */
```

```
/*************************************************************/
/*              Local Functions Declarations                 */
/*************************************************************/
static void error_print(struct ft_err *, char *);
static void exit_usr(int);

int main(int argc, char *argv[])
{
    char *ftstate;             /* Status string              */
    char *workdir;             /* Working directory          */
    int  rc;                   /* Some local return code     */
    char remotesys[201];       /* Address of remote systems  */
    char remoteadm[68];        /* Transfer admission         */
    char path[_MAX_PATH];      /* Path for working directory */
    char fud[STAT_FUD_LEN];    /* Further details            */
    struct stat buf;           /* stat() info of local file  */
    struct ft_err errorinfo;   /* Error information          */
    struct ft_admission admis; /* Admission parameters       */
    struct ft_transpar par;    /* File transfer parameters   */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Check program arguments                               */
    if (argc != 3)
    {
        printf("Call: %s <local file> <remote file>\n",
               argv[0]);
        return(1);
    }

    /* Check whether the local file exists                   */
    if (stat(argv[1], &buf) == -1)
    {
        printf("Error: Local file %s does not exist!\n",
               argv[1]);
        return(1);
    }

    /* Initialize ft_transpar structure                      */
    memset(&par, 0, sizeof(par));
    /* Set version of the parameter list                     */
    par.ftparvers = FT_TPARV2;
    /* Set direction to send a file to the remote system     */
    par.direction = FT_SEND;
    /* Set name of the local file to be sent                 */
    par.locfn = argv[1];
```

```
/* Set the name of the remote file                          */
par.remfn = argv[2];
/* Set file type to text mode                               */
par.filetype = FT_TEXT;
/* Set write mode to overwrite existing file                */
par.writemode = FT_OVERWR;
/* Set compression to Zip compatible compression mode       */
par.compress = FT_COMPRESSZIP;
/* Set name of local CCS, e.g. the contents of the local */
/* file consists of characters from the CP1252              */
/* character set                                            */
par.locccsn = "CP1252";
/* Set name of remote CCS, e.g. the contents of the         */
/* remote file will be coded corresponding to the UTF16  */
/* character set                                            */
par.remccsn = "UTF16";
/* Set buffer and buffer length for the further details  */
par.fud = fud;
par.fudlen = sizeof(fud);

/* Get the name/address of the remote system and the        */
/* transfer admission                                       */
printf("Enter name of remote system: ");
scanf("%s", remotesys);
printf("Enter transfer admission for remote system: ");
scanf("%s", remoteadm);
admis.remsys = remotesys;
admis.remadmis = remoteadm;
admis.remaccount = NULL;
admis.rempasswd = NULL;

/* Store pathname $TMP\ft                                   */
workdir = getenv("TMP");
strcpy(path, workdir);
strcat(path, "\\ft");

/* Prepare the options structure                            */
opt.ftoptsvers = FT_OPTSV1;
opt.ftapivers = FT_APIV2;

/* Open the API session                                     */
session_no = ft_open(path, &errorinfo, &opt);
if (session_no == NULL)
{
    /* Error                                                */
    error_print(&errorinfo, NULL);
}
```

```
/* Start the file transfer                                    */
req_id = ft_transfer(session_no, &admis, &par,
                     &errorinfo, &opt);
if (req_id == 0)
{
    /* Error                                                  */
    error_print(&errorinfo, par.fud);
}
printf("Transfer request started!\n");

/* Install signal handler to catch CTRL-C interrupts          */
signal(SIGINT, exit_usr);

/* Set the version of the ft_status structure                 */
state.ftstatvers = FT_STATV2;

/* Query the status of the request until the request is       */
/* terminated or aborted                                      */
do
{
    rc = ft_reqstat(session_no, req_id, &state,
                    &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                              */
        error_print(&errorinfo, NULL);
    }
    ftstate = "";
    switch (state.status)
    {
        case FT_STATW: ftstate = "Waiting";
                       break;
        case FT_STATR: ftstate = "Running";
                       break;
        case FT_STATA: ftstate = "Aborted";
                       break;
        case FT_STATT: ftstate = "Terminated";
                       break;
    }
    /* In the meanwhile do some other work...                 */
    Sleep(500);
```

```
    } while (state.status != FT_STATA &&
             state.status != FT_STATT);
    printf("Status of transfer request: %s\n", ftstate);
    if (state.status == FT_STATA)
        printf("Further details: %s\n", state.fud);

    /* Finish completed request                                  */
    rc = ft_reqterm(session_no, req_id, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                 */
        error_print(&errorinfo, NULL);
    }

    /* Close session                                             */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                 */
        error_print(&errorinfo, NULL);
    }
    return(0);
}

/*************************************************************/
/*************************************************************/
/*                                                           */
/*                    Local Functions                        */
/*                                                           */
/*************************************************************/
/*************************************************************/

/*************************************************************/
/* Simple error handling function                            */
/*************************************************************/
static void error_print(struct ft_err *errorinfo, char *fud)
{
    /* The meaning of the error codes are described in the   */
    /* include file ftapi.h and can be used for detailed     */
    /* error analysis.                                       */
    printf("errorinfo.main: %ld\n", errorinfo->main);
    printf("errorinfo.detail: %ld\n", errorinfo->detail);
    printf("errorinfo.additional: %ld\n",
            errorinfo->additional);
    if (fud != NULL)
        printf("further details: %s\n", fud);
    exit(EXIT_FAILURE);
}
```

```
/*************************************************************/
/* Signal handling function for SIGINT (CTRL-C)              */
/*************************************************************/
static void exit_usr(int s)
{
    int    rc;                 /* Some local return code     */
    char   *ftstate;           /* Status string              */
    struct ft_err errorinfo;   /* Information about error    */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Prepare the options structure                         */
    opt.ftoptsvers = FT_OPTSV1;
    opt.ftapivers = FT_APIV2;

    /* Set the version of the ft_status structure            */
    state.ftstatvers = FT_STATV2;

    /* Query the status of the request                       */
    rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                    &opt);
    if (rc == -1)
    {
        /* Error                                             */
        error_print(&errorinfo, NULL);
    }

    /* Abort the request                                     */
    if (state.status == FT_STATW || state.status == FT_STATR)
    {
        rc = ft_cancel(session_no, req_id, &errorinfo, &opt);
        if (rc == -1)
        {
            /* Error                                         */
            error_print(&errorinfo, NULL);
        }
        Sleep(3000);

        /* Query the status of the request                   */
        rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                        &opt);
        if (rc == -1)
        {
            /* Error                                         */
            error_print(&errorinfo, NULL);
        }
    }
```

```
    ftstate = "";
    switch (state.status)
    {
        case FT_STATW: ftstate = "Waiting";
                       break;
        case FT_STATR: ftstate = "Running";
                       break;
        case FT_STATA: ftstate = "Aborted";
                       break;
        case FT_STATT: ftstate = "Terminated";
                       break;
    }

    /* Finish request                                              */
    rc = ft_reqterm(session_no, req_id, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    printf("CTRL-C handled: FT request aborted!\n");
    printf("Status of request: %s\n", ftstate);
    if (state.status == FT_STATA)
        printf("Further details: %s\n", state.fud);

    /* Close session                                               */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    exit(EXIT_SUCCESS);
}
```

Unix systems

```
/*************************************************************/
/* sample1.c     File transfer with standard parameters      */
/*************************************************************/
/*                                                           */
/* Component Name  : sample1                                 */
/*                                                           */
/* Subcomponent of : SAMPLES                                 */
/*                                                           */
/* Systemdependency: CS-C-LIB (x)                            */
/*                                                           */
/* Copyright (c) Fujitsu Technology Solutions 2009           */
/*               All Rights Reserved                         */
/*                                                           */
/*************************************************************/
static const char Sccsid[] = "@(#)sample1.c   2.33 2009/03/25";
/*************************************************************

  Program call: sample1 <local file> <remote file>

 *************************************************************/
/*   Include Files                                           */
/*************************************************************/
#include <stdio.h>              /* printf()                  */
#include <stdlib.h>             /* exit()                    */
#include <string.h>             /* strcat(), strcpy()        */
#include <sys/types.h>          /* stat()                    */
#include <sys/stat.h>           /* stat()                    */
#include <unistd.h>             /* sleep()                   */
#include <signal.h>             /* sigaction(), sigemptyset() */
#include <ftapi.h>
/*FF*/

/*************************************************************/
/*                    Global Variables                       */
/*************************************************************/
void *session_no;    /* Session identification               */
long  req_id;        /* Request ID                           */

/*************************************************************/
/*               Local Functions Declarations                */
/*************************************************************/
static void error_print(struct ft_err *, char *);
static void exit_usr(int);
```

```
int main(int argc, char *argv[])
{
    struct sigaction sa;       /* structure for sigaction()  */
    char *ftstate;             /* Status string              */
    char *home;                /* Home directory             */
    int  rc;                   /* Some local return code     */
    char remotesys[201];       /* Address of remote systems  */
    char remoteadm[68];        /* Transfer admission         */
    char path[513];            /* Current directory          */
    char fud[STAT_FUD_LEN];    /* Further details            */
    struct stat buf;           /* stat() info of local file  */
    struct ft_err errorinfo;   /* Error information          */
    struct ft_admission admis; /* Admission parameters       */
    struct ft_transpar par;    /* File transfer parameters   */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Check program arguments                               */
    if (argc != 3)
    {
        printf("Call: %s <local file> <remote file>\n",
               argv[0]);
        return(1);
    }

    /* Check whether the local file exists                   */
    if (stat(argv[1], &buf) == -1)
    {
        printf("Error: Local file %s does not exist!\n",
               argv[1]);
        return(1);
    }

    /* Initialize ft_transpar structure                      */
    memset(&par, 0, sizeof(par));
    /* Set version of the parameter list                     */
    par.ftparvers = FT_TPARV2;
   /* Set direction to send a file to the remote system    */
    par.direction = FT_SEND;
    /* Set name of the local file to be sent                 */
    par.locfn = argv[1];
    /* Set the name of the remote file                       */
    par.remfn = argv[2];
    /* Set file type to text mode                            */
    par.filetype = FT_TEXT;
    /* Set write mode to overwrite existing file             */
    par.writemode = FT_OVERWR;
```

```
/* Set compression to Zip compatible compression mode    */
par.compress = FT_COMPRESSZIP;
/* Set name of local CCS, e.g. the contents of the local */
/* file consists of characters from the ISO8859-1        */
/* character set                                         */
par.locccsn = "ISO88591";
/* Set name of remote CCS, e.g. the contents of the      */
/* remote file will be coded corresponding to the        */
/* UTF16 character set                                   */
par.remccsn = "UTF16";
/* Set buffer and buffer length for the further details  */
par.fud = fud;
par.fudlen = sizeof(fud);

/* Get the name/address of the remote system and the     */
/* transfer admission                                    */
printf("Enter name of remote system: ");
scanf("%s", remotesys);
printf("Enter transfer admission for remote system: ");
scanf("%s", remoteadm);
admis.remsys     = remotesys;
admis.remadmis   = remoteadm;
admis.remaccount = NULL;
admis.rempasswd  = NULL;

/* Store pathname $HOME\ft                               */
home = getenv("HOME");
strcpy(path, home);
strcat(path, "/ft");

/* Prepare the options structure                         */
opt.ftoptsvers = FT_OPTSV1;
opt.ftapivers = FT_APIV2;

/* Open the API session                                  */
session_no = ft_open(path, &errorinfo, &opt);
if (session_no == NULL)
{
    /* Error                                             */
    error_print(&errorinfo, NULL);
}
```

```
/* Start the file transfer                                    */
req_id = ft_transfer(session_no, &admis, &par,
                     &errorinfo, &opt);
if (req_id == 0)
{
    /* Error                                                  */
    error_print(&errorinfo, par.fud);
}
printf("Transfer request started!\n");

/* Install signal handler for user abort                      */
sa.sa_handler = exit_usr;
sigemptyset(&sa.sa_mask);
sigaction(SIGINT, &sa, NULL);

/* Set the version of the ft_status structure                 */
state.ftstatvers = FT_STATV2;

/* Query the status of the request until the request is       */
/* terminated or aborted                                      */
do
{
    rc = ft_reqstat(session_no, req_id, &state,
                    &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                              */
        error_print(&errorinfo, NULL);
    }
    ftstate = "";
    switch (state.status)
    {
        case FT_STATW: ftstate = "Waiting";
                       break;
        case FT_STATR: ftstate = "Running";
                       break;
        case FT_STATA: ftstate = "Aborted";
                       break;
        case FT_STATT: ftstate = "Terminated";
                       break;
    }

    /* In the meanwhile some other work can be done...        */
    sleep(1);
```

```
    } while (state.status != FT_STATA &&
             state.status != FT_STATT);
    printf("Status of transfer request: %s\n", ftstate);
    if (state.status == FT_STATA)
        printf("Further details: %s\n", state.fud);

    /* Finish completed request                                  */
    rc = ft_reqterm(session_no, req_id, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                 */
        error_print(&errorinfo, NULL);
    }

    /* close the session                                         */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                 */
        error_print(&errorinfo, NULL);
    }
    return(0);
}

/*************************************************************/
/*************************************************************/
/*                                                           */
/*                    Local Functions                        */
/*                                                           */
/*************************************************************/
/*************************************************************/

/*************************************************************/
/* Simple error handling function                            */
/*************************************************************/
static void error_print(struct ft_err *errorinfo, char *fud)
{
    /* The meaning of the error codes are described in the   */
    /* include file ftapi.h and can be used for detailed     */
    /* error analysis.                                       */
    printf("errorinfo.main: %ld\n", errorinfo->main);
    printf("errorinfo.detail: %ld\n", errorinfo->detail);
    printf("errorinfo.additional: %ld\n",
            errorinfo->additional);
    if (fud != NULL)
        printf("further details: %s\n", fud);
    exit(EXIT_FAILURE);
}
```

```
/*************************************************************/
/* Signal handling function for SIGINT                       */
/*************************************************************/
static void exit_usr(int s)
{
    int    rc;                 /* Some local return code     */
    char   *ftstate;           /* Status string              */
    struct ft_err errorinfo;   /* Information about error    */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Prepare the options structure                         */
    opt.ftoptsvers = FT_OPTSV1;
    opt.ftapivers = FT_APIV2;

    /* Set the version of the ft_status structure            */
    state.ftstatvers = FT_STATV2;

    /* Query the status of the request                       */
    rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                    &opt);
    if (rc == -1)
    {
        /* Error                                             */
        error_print(&errorinfo, NULL);
    }

    /* Abort the request                                     */
    if (state.status == FT_STATW || state.status == FT_STATR)
    {
        rc = ft_cancel(session_no, req_id, &errorinfo, &opt);
        if (rc == -1)
        {
            /* Error                                         */
            error_print(&errorinfo, NULL);
        }
        sleep(3);

        /* Query the status of the request                   */
        rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                        &opt);
        if (rc == -1)
        {
            /* Error                                         */
            error_print(&errorinfo, NULL);
        }
    }
```

```
    ftstate = "";
    switch (state.status)
    {
        case FT_STATW: ftstate = "Waiting";
                       break;
        case FT_STATR: ftstate = "Running";
                       break;
        case FT_STATA: ftstate = "Aborted";
                       break;
        case FT_STATT: ftstate = "Terminated";
                       break;
    }

    /* Finish request                                              */
    rc = ft_reqterm(session_no, req_id, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    printf("SIGINT triggered: FT request aborted!\n");
    printf("Status of request: %s\n", ftstate);
    if (state.status == FT_STATA)
        printf("Further details: %s\n", state.fud);

    /* Close session                                               */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    exit(EXIT_SUCCESS);
}
```

Sample 2: Several file transfer requests with follow-up processing

The following program *sample2* is called in this way:

`sample2` *file1* [*file2*] [*file3*] [*file4*]

Name and transfer admission to the remote system are then queried in a dialog. For the program to run, the following directory must exist:

– Windows: the *ft* subdirectory in the variable *TMP*.
– Unix systems: the working directory `$HOME/ft`

The program fetches each of the files specified asynchronously from the HOME directory of the user of from the login name specified in the FT profile in the remote system. In the local system, the file is stored in the directory with the same name as the one from which the program was called. If file transfer has be completed successfully, the file is printed out in the local system. If the file was not transferred, the user is sent a message. If a SIGINT signal is created by the user (e.g by entering CTRL+C under Windows), while the file has not yet been transferred, the current file transfer request is aborted. No subsequent file transfer requests are initiated.

The program is structured as follows:

First, a session is first opened with the function *ft_open()*, where `%TMP%\ft` (under Windows) resp. `$HOME/ft` (under Unix systems) is permanently assigned as working directory. *ft_open()* returns a session number, which identifies the session and must be specified with further function calls.

The following procedure is repeated for all file to be transferred: The asynchronous file transfer is initiated with the *ft_transfer()* function. From now on, the program queries whether a SIGINT signal was created by the user. If so, the request is aborted with the *ft_cancel()* function, if the status is „Waiting“ or „Running“. As long a the file transfer has not been completed, the *ft_reqstat()* function queries the status of the file transfer request. If the status of the request is „Terminated“, follow-up processing is started, i.e.the file transferred to the local system is printed. If the status of the request is aborted „Aborted“, a message is output. In all cases, the file transfer request is marked as terminated with the *ft_reqterm()*.

The session is closed with the *ft_close()* function when all the file specified have been processed.

Windows

```
/*************************************************************/
/* sample2.c  Several FT requests with follow-up processing  */
/*************************************************************/
/*                                                           */
/* Component Name  : sample2                                 */
/*                                                           */
/* Subcomponent of : SAMPLES                                 */
/*                                                           */
/* Systemdependency: WINDOWS                                 */
/*                                                           */
/* Copyright (c) Fujitsu Technology Solutions 2009           */
/*               All Rights Reserved                         */
/*                                                           */
/*************************************************************/
static const char Sccsid[] = "@(#)sample1.c   2.33 2009/03/25";
/*************************************************************

  Program call: sample2 <remote file1>
                        [<remote file2>]
                        [<remote file3>]
                        [<remote file4>]

 *************************************************************/
/*                     Include Files                         */
/*************************************************************/
#include <windows.h>     /* Sleep()                          */
#include <stdio.h>       /* printf()                         */
#include <stdlib.h>      /* exit()                           */
#include <string.h>      /* strcat(), strcpy()               */
#include <signal.h>      /* signal()                         */
#include <ftapi.h>

/*************************************************************/
/*                    Global Variables                       */
/*************************************************************/
void *session_no;       /* Session identification            */
long  req_id;           /* Request ID                        */
```

```
/*************************************************************/
/*               Local Functions Declarations                */
/*************************************************************/
static void error_print(struct ft_err *, char *);
static void exit_usr(int);

int main (int argc, char *argv[])
{
    char *ftstate[5];          /* Status information         */
    char *workdir;             /* Working directory          */
    int  i;                    /* Some local counter         */
    int  rc;                   /* Some local return code     */
    int  done;                 /* Transfer terminated?       */
    char remotesys[201];       /* Address of remote system   */
    char remoteadm[68];        /* Transfer admission         */
    char path[_MAX_PATH];      /* Path for working directory */
    char fud[STAT_FUD_LEN];    /* Further details            */
    char command[_MAX_PATH];   /* Follow-up processing cmd   */
    struct ft_err errorinfo;   /* Error information          */
    struct ft_admission admis; /* Admission parameters       */
    struct ft_transpar par;    /* File transfer parameters   */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Check program arguments                               */
    if ((argc < 2) || (argc > 5))
    {
        printf("Call: %s <remote file1> [<remote file2>]"
               "[<remote file3>] [<remote file4>]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Initialize ft_transpar structure                      */
    memset(&par, 0, sizeof(par));
    /* Set version of the parameter list                     */
    par.ftparvers = FT_TPARV2;
    /* Set direction to fetch a file from the remote system  */
    par.direction = FT_RECEIVE;
    /* Set file type to text mode                            */
    par.filetype  = FT_TEXT;
    /* Set write mode to overwrite existing local file       */
    par.writemode = FT_OVERWR;
    /* Set compression to Zip compatible compression mode    */
    par.compress = FT_COMPRESSZIP;
    /* Set buffer and buffer length for the further details  */
    par.fud = fud;
    par.fudlen = sizeof(fud);
```

```
/* Get the name/address of the remote system and the     */
/* transfer admission                                    */
printf("Enter name of remote system: ");
scanf("%s", remotesys);
printf("Enter transfer admission for remote system: ");
scanf("%s", remoteadm);
admis.remsys = remotesys;
admis.remadmis = remoteadm;
admis.remaccount = NULL;
admis.rempasswd = NULL;

/* Store pathname $TMP\ft                                */
workdir = getenv("TMP");
strcpy(path, workdir);
strcat(path, "\\ft");

/* Prepare the options structure                         */
opt.ftoptsvers = FT_OPTSV1;
opt.ftapivers = FT_APIV2;

/* Open the API session                                  */
session_no = ft_open(path, &errorinfo, &opt);
if (session_no == NULL)
{
    /* Error                                             */
    error_print(&errorinfo, NULL);
}

/* Set the version of the ft_status structure            */
state.ftstatvers = FT_STATV2;

/* Start the file transfer with follow-up processing     */
for (i = 1; i < argc; i++)
{
    par.remfn = argv[i];
    par.locfn = argv[i];
    sprintf(command, "notepad.exe /p \"%s\"", argv[i]);
    par.locsuccproc = command;
    req_id = ft_transfer(session_no, &admis, &par,
                         &errorinfo, &opt);
    if (req_id == 0)
    {
        /* Error                                        */
        error_print(&errorinfo, par.fud);
    }
    printf("Transfer request %d started!\n", i);
```

```
/* Install signal handler to catch CTRL-C interrupts */
signal(SIGINT, exit_usr);

/* Query the status of the request until the request */
/* is terminated or aborted                          */
done = 0;
do
{
    rc = ft_reqstat(session_no, req_id, &state,
                    &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                     */
        error_print(&errorinfo, NULL);
    }
    ftstate[i] = "";
    switch (state.status)
    {
        case FT_STATW: ftstate[i] = "Waiting";
                       break;
        case FT_STATR: ftstate[i] = "Running";
                       break;
        case FT_STATA: ftstate[i] = "Aborted";
                       break;
        case FT_STATT: ftstate[i] = "Terminated";
                       break;
    }

    /* Check if request was terminated or aborted    */
    if (state.status == FT_STATA ||
        state.status == FT_STATT)
    {
        done = 1;
        /* Finish completed request                  */
        rc = ft_reqterm(session_no, req_id,
                        &errorinfo, &opt);
        if (rc == -1)
        {
            /* Error                                 */
            error_print(&errorinfo, NULL);
        }
        if (state.status == FT_STATA)
        {
            printf("Transfer of file %s failed!\n",
                   argv[i]);
            printf("Further details: %s\n", state.fud);
        }
    }
```

```
            /* In the meanwhile do some other work...        */
            Sleep(500);

        } while (!done);
    }

    /* Display status of requests                            */
    for (i = 1; i < argc; i++)
    {
        printf("Transfer of file %s ended in status %s.\n",
               argv[i], ftstate[i]);
    }

    /* close the session                                     */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                             */
        error_print(&errorinfo, NULL);
    }
    return(0);
}

/*************************************************************/
/*************************************************************/
/*                                                           */
/*                    Local Functions                        */
/*                                                           */
/*************************************************************/
/*************************************************************/

/*************************************************************/
/* Simple error handling function                            */
/*************************************************************/
static void error_print(struct ft_err *errorinfo, char *fud)
{
    /* The meaning of the error codes are described in the   */
    /* include file ftapi.h and can be used for detailed     */
    /* error analysis.                                       */
    printf("errorinfo.main: %ld\n", errorinfo->main);
    printf("errorinfo.detail: %ld\n", errorinfo->detail);
    printf("errorinfo.additional: %ld\n",
            errorinfo->additional);
    if (fud != NULL)
        printf("further details: %s\n", fud);
    exit(EXIT_FAILURE);
}
```

```
/*************************************************************/
/* Signal handling function for SIGINT (CTRL-C)              */
/*************************************************************/
static void exit_usr(int s)
{
    int    rc;                 /* Some local return code     */
    char   *ftstate;           /* Status string              */
    struct ft_err errorinfo;   /* Information about error    */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Prepare the options structure                         */
    opt.ftoptsvers = FT_OPTSV1;
    opt.ftapivers = FT_APIV2;

    /* Set the version of the ft_status structure            */
    state.ftstatvers = FT_STATV2;

    /* Query the status of the request                       */
    rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                    &opt);
    if (rc == -1)
    {
        /* Error                                             */
        error_print(&errorinfo, NULL);
    }

    /* Abort the request                                     */
    if (state.status == FT_STATW || state.status == FT_STATR)
    {
        rc = ft_cancel(session_no, req_id, &errorinfo, &opt);
        if (rc == -1)
        {
            /* Error                                         */
            error_print(&errorinfo, NULL);
        }
        Sleep(3000);

        /* Query the status of the request                   */
        rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                        &opt);
        if (rc == -1)
        {
            /* Error                                         */
            error_print(&errorinfo, NULL);
        }
    }
```

```
    ftstate = "";
    switch (state.status)
    {
        case FT_STATW: ftstate = "Waiting";
                       break;
        case FT_STATR: ftstate = "Running";
                       break;
        case FT_STATA: ftstate = "Aborted";
                       break;
        case FT_STATT: ftstate = "Terminated";
                       break;
    }

    /* Finish request                                              */
    rc = ft_reqterm(session_no, req_id, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    printf("CTRL-C handled: FT request aborted!\n");
    printf("Status of request: %s\n", ftstate);
    if (state.status == FT_STATA)
        printf("Further details: %s\n", state.fud);

    /* Close session                                               */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    exit(EXIT_SUCCESS);
}
```

Unix systems

```
/*sample2.c  Several FT requests with follow-up processing   */
/*************************************************************/
/*                                                           */
/* Component Name  : sample2                                 */
/*                                                           */
/* Subcomponent of : SAMPLES                                 */
/*                                                           */
/* Systemdependency: CS-C-LIB (x)                            */
/*                                                           */
/* Copyright (c) Fujitsu Technology Solutions 2009           */
/*               All Rights Reserved                         */
/*                                                           */
/*************************************************************/
static const char Sccsid[] = "@(#)sample1.c   2.33 2009/03/25";
/*************************************************************

  Program call: sample2 <remote file1>
                        [<remote file2>]
                        [<remote file3>]
                        [<remote file4>]

 *************************************************************/
/*                      Include Files                        */
/*************************************************************/
#include <stdio.h>       /* printf()                         */
#include <stdlib.h>      /* exit()                           */
#include <string.h>      /* strcat(), strcpy()               */
#include <unistd.h>      /* sleep()                          */
#include <signal.h>      /* sigaction(), sigemptyset()       */
#include <ftapi.h>

/*************************************************************/
/*                     Global Variables                      */
/*************************************************************/
void *session_no;       /* Session identification           */
long  req_id;           /* Request ID                       */

/*************************************************************/
/*               Local Functions Declarations                */
/*************************************************************/
static void error_print(struct ft_err *, char *);
static void exit_usr(int);

int main (int argc, char *argv[])
{
    struct sigaction sa;       /* structure for sigaction()  */
```

```
    char *ftstate[5];          /* Status information         */
    char *home;                /* Home directory             */
    int  i;                    /* Some local counter         */
    int  rc;                   /* Some local return code     */
    int  done;                 /* Transfer terminated?       */
    char remotesys[201];       /* Address of remote system   */
    char remoteadm[68];        /* Transfer admission         */
    char path[513];            /* Path for working directory */
    char command[500];         /* Follow-up processing cmd   */
    char fud[STAT_FUD_LEN];    /* Further details            */
    struct ft_err errorinfo;   /* Error information          */
    struct ft_admission admis; /* Admission parameters       */
    struct ft_transpar par;    /* File transfer parameters   */
    struct ft_status state;    /* Status information         */
    struct ft_options opt;     /* Options                    */

    /* Check program arguments                               */
    if ((argc < 2) || (argc > 5))
    {
        printf("Call: %s <remote file1> [<remote file2>]"
               "[<remote file3>] [<remote file4>]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Initialize ft_transpar structure                      */
    memset(&par, 0, sizeof(par));
    /* Set version of the parameter list                     */
    par.ftparvers = FT_TPARV2;
/* Set direction to fetch a file from the remote system */
    par.direction = FT_RECEIVE;
    /* Set file type to text mode                            */
    par.filetype  = FT_TEXT;
    /* Set write mode to overwrite existing local file       */
    par.writemode = FT_OVERWR;
/* Set compression to Zip compatible compression mode  */
    par.compress = FT_COMPRESSZIP;
    /* Set buffer and buffer length for the further details  */
    par.fud = fud;
    par.fudlen = sizeof(fud);

    /* Get the name/address of the remote system and the     */
    /* transfer admission                                    */
    printf("Enter name of remote system: ");
    scanf("%s", remotesys);
    printf("Enter transfer admission for remote system: ");
    scanf("%s", remoteadm);
    admis.remsys = remotesys;
    admis.remadmis = remoteadm;
```

```
admis.remaccount = NULL;
admis.rempasswd = NULL;

/* Store pathname $(HOME)/ft                                    */
home = getenv("HOME");
strcpy(path, home);
strcat(path, "/ft");

/* Prepare the options structure                                */
opt.ftoptsvers = FT_OPTSV1;
opt.ftapivers = FT_APIV2;

/* Open the API session                                         */
session_no = ft_open(path, &errorinfo, &opt);
if (session_no == NULL)
{
    /* Error                                                    */
    error_print(&errorinfo, NULL);
}

/* Set the version of the ft_status structure                   */
state.ftstatvers = FT_STATV2;

/* Start the file transfer with follow-up processing            */
for (i = 1; i < argc; i++)
{
    par.remfn = argv[i];
    par.locfn = argv[i];
    sprintf(command, "lpr %s", argv[i]);
    par.locsuccproc = command;
    req_id = ft_transfer(session_no, &admis, &par,
                         &errorinfo, &opt);
    if (req_id == 0)
    {
        /* Error                                                */
        error_print(&errorinfo, par.fud);
    }
    printf("Transfer request %d started!\n", i);

    /* Install signal handler for user abort                    */
    sa.sa_handler = exit_usr;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGINT, &sa, NULL);

    /* Query the status of the request until the request        */
    /* is terminated or aborted                                 */
    done = 0;
    do
```

```
    {
        rc = ft_reqstat(session_no, req_id, &state,
                        &errorinfo, &opt);
        if (rc == -1)
        {
            /* Error                                          */
            error_print(&errorinfo, NULL);
        }
        ftstate[i] = "";
        switch (state.status)
        {
            case FT_STATW: ftstate[i] = "Waiting";
                           break;
            case FT_STATR: ftstate[i] = "Running";
                           break;
            case FT_STATA: ftstate[i] = "Aborted";
                           break;
            case FT_STATT: ftstate[i] = "Terminated";
                           break;
        }

        /* Check if request was terminated or aborted    */
        if (state.status == FT_STATA ||
            state.status == FT_STATT)
        {
            done = 1;

            /* Finish completed request                   */
            rc = ft_reqterm(session_no, req_id,
                            &errorinfo, &opt);
            if (rc == -1)
            {
                /* Error                                  */
                error_print(&errorinfo, NULL);
            }
            if (state.status == FT_STATA)
            {
                printf("Transfer of file %s failed!\n",
                       argv[i]);
                printf("Further details: %s\n", state.fud);
            }
        }

        /* In the meanwhile do some other work...         */
        sleep(1);

    } while (!done);
}
```

```
    /* Display status of requests                                */
    for (i = 1; i < argc; i++)
    {
        printf("Transfer of file %s ended in status %s.\n",
               argv[i], ftstate[i]);
    }

    /* close the session                                         */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                 */
        error_print(&errorinfo, NULL);
    }
    return(0);
}


/*************************************************************/
/*************************************************************/
/*                                                           */
/*                    Local Functions                        */
/*                                                           */
/*************************************************************/
/*************************************************************/

/*************************************************************/
/* Simple error handling function                            */
/*************************************************************/
static void error_print(struct ft_err *errorinfo, char *fud)
{
    /* The meaning of the error codes are described in the   */
    /* include file ftapi.h and can be used for detailed     */
    /* error analysis.                                       */
    printf("errorinfo.main: %ld\n", errorinfo->main);
    printf("errorinfo.detail: %ld\n", errorinfo->detail);
    printf("errorinfo.additional: %ld\n",
            errorinfo->additional);
    if (fud != NULL)
        printf("further details: %s\n", fud);
    exit(EXIT_FAILURE);
}


/*************************************************************/
/* Signal handling function for SIGINT                       */
/*************************************************************/
```

```
static void exit_usr(int s)
{
    int    rc;                  /* Some local return code     */
    char   *ftstate;            /* Status string              */
    struct ft_err errorinfo;    /* Information about error    */
    struct ft_status state;     /* Status information         */
    struct ft_options opt;      /* Options                    */

    /* Prepare the options structure                          */
    opt.ftoptsvers = FT_OPTSV1;
    opt.ftapivers = FT_APIV2;

    /* Set the version of the ft_status structure             */
    state.ftstatvers = FT_STATV2;

    /* Query the status of the request                        */
    rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                    &opt);
    if (rc == -1)
    {
        /* Error                                              */
        error_print(&errorinfo, NULL);
    }

    /* Abort the request                                      */
    if (state.status == FT_STATW || state.status == FT_STATR)
    {
        rc = ft_cancel(session_no, req_id, &errorinfo, &opt);
        if (rc == -1)
        {
            /* Error                                          */
            error_print(&errorinfo, NULL);
        }
        sleep(3);
        /* Query the status of the request                    */
        rc = ft_reqstat(session_no, req_id, &state, &errorinfo,
                        &opt);
        if (rc == -1)
        {
            /* Error                                          */
            error_print(&errorinfo, NULL);
        }
    }
    ftstate = "";
    switch (state.status)
    {
        case FT_STATW: ftstate = "Waiting";
                       break;
```

```
        case FT_STATR: ftstate = "Running";
                       break;
        case FT_STATA: ftstate = "Aborted";
                       break;
        case FT_STATT: ftstate = "Terminated";
                       break;
    }

    /* Finish request                                              */
    rc = ft_reqterm(session_no, req_id, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    printf("SIGINT triggered: FT request aborted!\n");
    printf("Status of request: %s\n", ftstate);
    if (state.status == FT_STATA)
        printf("Further details: %s\n", state.fud);

    /* Close session                                               */
    rc = ft_close(session_no, &errorinfo, &opt);
    if (rc == -1)
    {
        /* Error                                                   */
        error_print(&errorinfo, NULL);
    }
    exit(EXIT_SUCCESS);
}
```

Sample 3: Display contents of a directory in a remote system

The following program *sample3* is called in this way:

```
sample3 dvz1
```

Name and transfer admission to the remote system are then queried in a dialog.

The program reads the entries in directory *dir1* of the remote system and outputs the list on screen. The directory must be specified as an absolute path name, or as relative to the HOME directory (see page 10) of the user or login name specified in the FT profile in the remote system. Up to 10 information entries are output in thi example, even if the directory specified contains more than 10 files/directories.

The program is structured as follows:

The *ft_showdir()* function reads the information relating to the specified directory in the remote system. A buffer is provided for this purpose, which is capable of holding the information on a total of 10 files or directories. The number of entries is also supplied.

Windows

```
/*************************************************************/
/* sample3.c  File management requests                       */
/*************************************************************/
/*                                                           */
/* Component Name  : sample3                                 */
/*                                                           */
/* Subcomponent of : SAMPLES                                 */
/*                                                           */
/* Systemdependency: WINDOWS                                 */
/*                                                           */
/* Copyright (c) Fujitsu Technology Solutions 2009           */
/*               All Rights Reserved                         */
/*                                                           */
/*************************************************************/
static const char Sccsid[] = "@(#)sample1.c   2.33 2009/03/25";
/*************************************************************

  Program call: sample3 <directory>
```

```
 ************************************************************/
/*   Include Files                                          */
/************************************************************/
#include <stdio.h>       /* printf()                        */
#include <stdlib.h>      /* exit()                          */
#include <string.h>      /* strcat(), strcpy()              */
#include <ftapi.h>

/************************************************************/
/*                 Local Constants and Macros               */
/************************************************************/
#define BUFSIZE 200      /* number of ft_fileinfo structures */
                         /* in output buffer                */

/************************************************************/
/*               Local Functions Declarations               */
/************************************************************/
static void error_print(struct ft_err *, char *);
static void printinfo(struct ft_fileinfo *info);

int main (int argc, char *argv[])
{
    int i;                      /* Some local counter       */
    int count;                  /* Number of files          */
    char remotesys[201];        /* Address of remote systems  */
    char remoteadm[68];         /* Transfer admission       */
    char fud[STAT_FUD_LEN];     /* Further details          */
    struct ft_err errorinfo;    /* Error information        */
    struct ft_admission admis; /* Admission parameters      */
    struct ft_shwpar par;       /* Show directory parameters  */
    struct ft_options opt;      /* Options                  */
    struct ft_fileinfo buf[BUFSIZE]; /* Output buffer       */

    /* Check program arguments                              */
    if (argc != 2)
    {
        printf("Call: %s <directory>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Initialize ft_transpar structure                     */
    memset(&par, 0, sizeof(par));
    /* Set version of the parameter list                    */
    par.shwparvers  = FT_SPARV2;
    /* Set the name of the remote directory                 */
    par.fn = argv[1];
    /* Set buffer and buffer length for the further details  */
```

```
    par.fud = fud;
    par.fudlen = sizeof(fud);
    /* Set version of the ft_fileinfo structure in the       */
    /* output buffer. The version in the first structure     */
    /* is valid for all structtures in the output buffer.    */
    buf[0].ftshowivers = FT_SHOWIV2;

    /* Get the name/address of the remote system and the     */
    /* transfer admission                                    */
    printf("Enter name of remote system: ");
    scanf("%s", remotesys);
    printf("Input transfer admission to remote system: ");
    scanf("%s", remoteadm);
    admis.remsys     = remotesys;
    admis.remadmis   = remoteadm;
    admis.remaccount = NULL;
    admis.rempasswd  = NULL;

    /* Prepare the options structure                         */
    opt.ftoptsvers = FT_OPTSV1;
    opt.ftapivers = FT_APIV2;

    /* Read the contents of the remote directory             */
    count = ft_showdir(&admis, &par, buf, BUFSIZE, &errorinfo,
                       &opt);
    if (count == -1)
    {
        /* Error                                             */
        error_print(&errorinfo, par.fud);
    }

    /* Display result of request                             */
    printf("There are %d entries in remote directory %s\n",
           count, argv[1]);

    /* If the output range was not large enough for all      */
    /* the information, only the data in the buffer is       */
    /* displayed                                             */
    if (count > BUFSIZE)
        count = BUFSIZE;
    for (i = 0; i < count; i++)
         printinfo(&buf[i]);
    return(0);
}
```

```
/*************************************************************/
/*************************************************************/
/*                                                           */
/*                    Local Functions                        */
/*                                                           */
/*************************************************************/
/*************************************************************/

/*************************************************************/
/* Simple error handling function                            */
/*************************************************************/
static void error_print(struct ft_err *errorinfo, char *fud)
{
    /* The meaning of the error codes are described in the   */
    /* include file ftapi.h and can be used for detailed     */
    /* error analysis.                                       */
    printf("errorinfo.main: %ld\n", errorinfo->main);
    printf("errorinfo.detail: %ld\n", errorinfo->detail);
    printf("errorinfo.additional: %ld\n",
            errorinfo->additional);
    if (fud != NULL)
        printf("further details: %s\n", fud);
    exit(EXIT_FAILURE);
}

/*************************************************************/
/* Display information of a file in the remote system        */
/*************************************************************/
static void printinfo(struct ft_fileinfo *info)
{
printf("File name: %s\n", info->fn);
printf("File type: ");
switch (info->filetype)
  {
  case FT_BIN: printf("Binary\n"); break;
  case FT_DIR: printf("Directory\n"); break;
  case FT_TXT: printf("Text\n"); break;
  case FT_NOTYPE: printf("Unknown\n"); break;
  }
printf("Character set: ");
switch (info->charset)
  {
  case FT_NOSET:   printf("Unknown\n"); break;
  case FT_GRAPHIC: printf("GraphicString\n"); break;
  case FT_GENERAL: printf("GeneralString\n"); break;
  case FT_IA5:     printf("IA5String\n"); break;
  case FT_VISIBLE: printf("VisibleString\n"); break;
  }
```

```
printf("Recordform: ");
switch (info->recordform)
  {
  case FT_NOFORM:   printf("Unknown\n"); break;
  case FT_VARIABLE: printf("Variable length\n"); break;
  case FT_FIXED:    printf("Fixed length\n"); break;
  case FT_UNDEF:    printf("Undefined record length\n"); break;
  }
printf("Max. record length: %ld\n", info->recsize);
printf("Availability: ");
switch (info->availability)
  {
  case FT_NOAVAIL:   printf("Unknown\n"); break;
  case FT_AVAILIMM:  printf("Immediate\n"); break;
  case FT_AVAILNIMM: printf("Not immediate\n"); break;
  }
printf("Access rights: %d\n", info->access);
printf("Account number.: %s\n", info->account);
printf("Legal qualif.: %s\n", info->legalqual);
printf("Create: %s %ld\n", info->cre_user, info->cre_date);
printf("Modification: %s %ld\n", info->mod_user,
       info->mod_date);
printf("Read: %s %ld\n", info->rea_user, info->rea_date);
printf("Modify attribute: %s %ld\n", info->atm_user,
       info->atm_date);
printf("File size: %I64d\n", info->fsize);
printf("Max. size: %I64d\n", info->fmaxsize);
printf("---\n");
}
```

Unix systems

```
/*************************************************************/
/* sample3.c     File management requests                    */
/*************************************************************/
/*                                                           */
/* Component Name  : sample3                                 */
/*                                                           */
/* Subcomponent of : SAMPLES                                 */
/*                                                           */
/* Systemdependency: CS-C-LIB (x)                            */
/*                                                           */
/* Copyright (c) Fujitsu Technology Solutions 2009           */
/*               All Rights Reserved                         */
/*                                                           */
/*************************************************************/
static const char Sccsid[] = "@(#)sample1.c   2.33 2009/03/25";
/*************************************************************

  Program call: sample3 <directory>

 *************************************************************/
/*   Include Files                                           */
/*************************************************************/
#include <stdio.h>       /* printf()                         */
#include <stdlib.h>      /* exit()                           */
#include <string.h>      /* strcat(), strcpy()               */
#include <ftapi.h>

/*************************************************************/
/*                 Local Constants and Macros                */
/*************************************************************/
#define BUFSIZE 200      /* number of ft_fileinfo structures */
                         /* in output buffer                 */

/*************************************************************/
/*               Local Functions Declarations                */
/*************************************************************/
static void error_print(struct ft_err *, char *);
static void printinfo(struct ft_fileinfo *info);
```

```
int main (int argc, char *argv[])
{
    int i;                      /* Some local counter         */
    int count;                  /* Number of files            */
    char remotesys[201];        /* Address of remote systems  */
    char remoteadm[68];         /* Transfer admission         */
    char fud[STAT_FUD_LEN];     /* Further details            */
    struct ft_err errorinfo;    /* Error information          */
    struct ft_admission admis; /* Admission parameters        */
    struct ft_shwpar par;       /* Show directory parameters  */
    struct ft_options opt;      /* Options                    */
    struct ft_fileinfo buf[BUFSIZE]; /* Output buffer         */

    /* Check program arguments                                */
    if (argc != 2)
    {
        printf("Call: %s <directory>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Initialize ft_transpar structure                       */
    memset(&par, 0, sizeof(par));
    /* Set version of the parameter list                      */
    par.shwparvers  = FT_SPARV2;
    /* Set the name of the remote directory                   */
    par.fn = argv[1];
    /* Set buffer and buffer length for the further details   */
    par.fud = fud;
    par.fudlen = sizeof(fud);
    /* Set version of the ft_fileinfo structure in the        */
    /* output buffer. The version in the first structure      */
    /* is valid for all structtures in the output buffer.     */
    buf[0].ftshowivers = FT_SHOWIV2;

    /* Get the name/address of the remote system and the      */
    /* transfer admission                                     */
    printf("Enter name of remote system: ");
    scanf("%s", remotesys);
    printf("Input transfer admission to remote system: ");
    scanf("%s", remoteadm);
    admis.remsys     = remotesys;
    admis.remadmis   = remoteadm;
    admis.remaccount = NULL;
    admis.rempasswd  = NULL;

    /* Prepare the options structure                          */
    opt.ftoptsvers = FT_OPTSV1;
    opt.ftapivers = FT_APIV2;
```

```
    /* Read the contents of the remote directory            */
    count = ft_showdir(&admis, &par, buf, BUFSIZE, &errorinfo,
                       &opt);
    if (count == -1)
    {
        /* Error                                            */
        error_print(&errorinfo, par.fud);
    }

    /* Display result of request                            */
    printf("There are %d entries in remote directory %s\n",
           count, argv[1]);

    /* If the output range was not large enough for all     */
    /* the information, only the data in the buffer is      */
    /* displayed                                            */
    if (count > BUFSIZE)
        count = BUFSIZE;
    for (i = 0; i < count; i++)
         printinfo(&buf[i]);
    return(0);
}

/*************************************************************/
/*************************************************************/
/*                                                           */
/*                    Local Functions                        */
/*                                                           */
/*************************************************************/
/*************************************************************/

/*************************************************************/
/* Simple error handling function                            */
/*************************************************************/
static void error_print(struct ft_err *errorinfo, char *fud)
{
    /* The meaning of the error codes are described in the   */
    /* include file ftapi.h and can be used for detailed     */
    /* error analysis.                                       */
    printf("errorinfo.main: %ld\n", errorinfo->main);
    printf("errorinfo.detail: %ld\n", errorinfo->detail);
    printf("errorinfo.additional: %ld\n",
            errorinfo->additional);
    if (fud != NULL)
        printf("further details: %s\n", fud);
    exit(EXIT_FAILURE);
}
```

```
/*************************************************************/
/* Display information of a file in the remote system        */
/*************************************************************/
static void printinfo(struct ft_fileinfo *info)
{
printf("File name: %s\n", info->fn);
printf("File type: ");
switch (info->filetype)
  {
  case FT_BIN: printf("Binary\n"); break;
  case FT_DIR: printf("Directory\n"); break;
  case FT_TXT: printf("Text\n"); break;
  case FT_NOTYPE: printf("Unknown\n"); break;
  }
printf("Character set: ");
switch (info->charset)
  {
  case FT_NOSET:   printf("Unknown\n"); break;
  case FT_GRAPHIC: printf("GraphicString\n"); break;
  case FT_GENERAL: printf("GeneralString\n"); break;
  case FT_IA5:     printf("IA5String\n"); break;
  case FT_VISIBLE: printf("VisibleString\n"); break;
  }
printf("Recordform: ");
switch (info->recordform)
  {
  case FT_NOFORM:   printf("Unknown\n"); break;
  case FT_VARIABLE: printf("Variable length\n"); break;
  case FT_FIXED:    printf("Fixed length\n"); break;
  case FT_UNDEF:    printf("Undefined record length\n");
                    break;
  }
printf("Max. record length: %ld\n", info->recsize);
printf("Availability: ");
switch (info->availability)
  {
  case FT_NOAVAIL:   printf("Unknown\n"); break;
  case FT_AVAILIMM:  printf("Immediate\n"); break;
  case FT_AVAILNIMM: printf("Not immediate\n"); break;
  }
printf("Access rights: %d\n", info->access);
printf("Account number.: %s\n", info->account);
printf("Legal qualif.: %s\n", info->legalqual);
printf("Create: %s %ld\n", info->cre_user, info->cre_date);
printf("Modification: %s %ld\n", info->mod_user,
       info->mod_date);
```

```
printf("Read: %s %ld\n", info->rea_user, info->rea_date);
printf("Modify attribute: %s %ld\n", info->atm_user,
       info->atm_date);
printf("File size: %lld\n", info->fsize);
printf("Max. size: %lld\n", info->fmaxsize);
printf("---\n");
}
```

Index

%ELEMNAME 60
%ELEMTYPE 60
%ELEMVERS 60
%FILENAME 59
%PARTNER 59
%RESULT 59
%RID 60

A
additional error message
program interface 15
address
partner system 13
API 5
asynchronous file transfer
program interface 7
asynchronous request
manage (API) 11, 20
attributes
of remote file (API) 40
several remote files (API) 46
automate
sequences 5

C
cancel
request (API) 20
CCS name
local 64
remote 64
close
session (API) 22
coding
remote 64
codong
local 64
contents
remote directory (API) 46
create
directory in remote system (prog. int.) 23

D
delete
directory in remote system (API) 25
file in remote system (API) 25
management information (API) 38
request (API) 20
detail error message 15
directory
contents of a remote (API) 46
creating in remot e system (prog. int.) 23
delete in remote system (API) 25

E
error class
program interface 67
error handling
program interface 15
error message
program interface 15, 67

F
fetch file
program interface 53
file
asynchronous transfer (API) 7
attributes in remote system (API) 46
delete in remote system (API) 25
fetch (API) 53
send (API) 53
synchronous transfer (API) 7, 53
transfer (API) 6
file attributes
in the remote system (API) 40

file management
program interface 6
file transfer request
manage 6
ft_admission 12
ft_cancel 20
ft_close 22
ft_credir 23
ft_open 28
ft_properties 30
ft_reqlist 33
ft_reqstat 35
ft_reqterm 38
ft_show 40
ft_showdir 46
ft_transfer 53
FTAM protocol 13
ftp protocol 13
function calls
program interface 6

H
header file 12, 15
HOME directory
Windows 10
host name 13

I
include file 17
input parameters
program interface 19

L
linking
program 18

M
main
error message 15
manage
transfer request 6
transfer request (API) 11, 20
management information
delete (API) 38
multithreading 5

O
open
session (API) 28
openFT protocol 13
output parameters
program interface 19

P
parameters
program interface 19
partner list 13
presentation selector 13
program
translate 18
program interface 5
determine version 30
function calls 6
query properties 6
program structure (API) 8
programming rules 7

Q
query properties
program interface 6

R
record format 65
remote system
admission 12
Request
delete (API) 20
request
cancel (API) 20
determine (API) 35
program interface 7
terminate (API) 38
request ID 9

return value
on error (API) 15
rules
for program interface 7

S
send
file (program interface) 53
sequence
automate 5
session
close (API) 22
open (API) 28
program interface 7
session identification 8
session number 8, 28
session selector 13
shell variable 60
status
of a request (API) 35, 36
synchronous file transfer
program interface 7
system value
version-specific 30

T
tabulator expansion 64
terminate
request (API) 38
thread 5
threadsafe 5
transfer file
asynchronous (API) 7, 53
program interface 6
synchronous (API) 7, 53
translate
program 18
transport selector 13

V
version
determine for API 16
program interface 30
version-specific system value
program interface 16, 30

W
working directory 28
program interface 8